3

BRUCKMANN - ENGLISCH
GERITS - WALKOWIAK

MICRO APPLICATION

TRUCS ET ASTUCES DE L'ATARI ST

ATARIST

UN LIVRE DATA BECKER

3

BRUCKMANN - ENGLISCH
GERITS - WALKOWIAK

MICRO APPLICATION

TRUCS ET ASTUCES DE L'ATARI ST

ATARI ST

UN LIVRE DATA BECKER

Distribué par : MICRO APPLICATION
13, Rue Sainte Cécile
75009 PARIS

et

EDITION RADIO
9, Rue Jacob
75006 PARIS

(c) Reproduction interdite sans l'autorisation de
MICRO APPLICATION

'Toute représentation ou reproduction, intégrale ou partielle, faite sans le consentement de MICRO APPLICATION est illicite (Loi du 11 Mars 1957, article 40, 1er alinéa).

Cette représentation ou reproduction illicite, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles 425 et suivants de Code Pénal.

La Loi du 11 Mars 1957 n'autorise, aux termes des alinéas 2 et 3 de l'article 41, que les copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à l'utilisation collective d'une part, et d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration'.

ISBN : 2-86899-043-6

(c) 1985 DATA BECKER
Merowingerstrasse, 30
4000 DUSSELDORF
R.F.A.

Traduction Française assurée par Olivier POLETTE

(c) 1985 MICRO APPLICATION
13 Rue Sainte Cécile
75009 PARIS

Collection dirigée par Mr Philippe OLIVIER

SOMMAIRE

1. LE BASIC DE L'ATARI 1

1.1. Les instructions BASIC les plus intéressantes. 1

1.2. BASIC et GEM 23

1.2.1. L'instruction VDISYS 24
1.2.2. Routines utiles du VDI en BASIC 27
1.2.3. L'instruction GEMSYS 42

1.3. La rapidité du BASIC 44

1.4. BASIC et langage machine 46

1.4.1. Les endroits pour le langage machine 46

1.5. L'horloge la plus chère de votre foyer 51

1.6. HARDCOPY automatique 57

2. UTILITAIRES POUR L'ATARI ST 58

2.1. L'heure à tout moment 58

2.2. driver d'imprimante pour l'Epson 68

2.3. Disque virtuel en RAM 74

2.4. Spooler d'imprimante 86

2.5. Mise en route automatique des applications du TOS 96

2.6. C et le langage machine 101

3. HARDCOPY EN COULEUR 113

3.1. Connection d'un écran couleur 114

3.2. Structure de la RAM graphique 117

3.3. HARDCOPY 123

3.3.1. Imprimante matricielle 124
3.3.2. Traceur 153

4. L'ENVIRONNEMENT GEM 172

4.1.1. Anatomie de GEM 173
4.1.2. Le VDI 174
4.1.3. L'AES 176
4.1.4. Le resource-file 181
4.1.5. La programmation sous TOS 181

4.2. Le jeu du 21 182

4.3. Le pas suivant : une application avec GEM 189

4.3.1. PRINIT- Un exemple d'application 207

4.4. Construction d'un fichier RSC 209

4.5. PRINIT en tant qu'accessory 227

CHAPITRE 1

LE BASIC DE L'ATARI

Le BASIC ATARI ainsi que le langage LOGO sont livrés avec l'ATARI ST. Il est bien connu que la partie matériel d'un nouvel ordinateur est toujours prête bien avant que les logiciels nécessaires ne soient terminés. Le BASIC du ST ne faillit pas à cette règle.
Jusqu'ici, seules quelques pré-versions étaient disponibles chez les vendeurs. Cependant, nous pouvons vous assurer que les dernières versions existantes à la sortie de cet ouvrage n'on presque plus d'erreurs. Beaucoup d'aspects du BASIC actuel sont ainsi construits et l'on peut prévoir qu'ils ne seront pas transformés dans des versions ultérieures. Il n'est donc pas inutile de parler de ces points et de les utiliser avec le langage machine. Au pire le BASIC s'améliorera (et prendra peut-être moins de place ?).
Si vous avez déjà programmé en BASIC sur d'autres machines, vous vous adapterez sans doute facilement au BASIC du ST. Le BASIC développé par DIGITAL RESEARCH se rapproche syntaxiquement du BASIC MICROSOFT. On peut le comparer à celui de l'IBM PC. Seules quelques spécificités dans la structure de l'IBM justifient des instructions qui ne sont pas disponibles avec le BASIC du ST. D'autre part, les programmeurs en BASIC ST disposent de possibilités dont les possesseurs d'autres versions ne peuvent que rêver.
Ainsi, il existe un interfaçage très souple avec GEM, ou plus exactement avec le VDI. Le VDI possède des possibilités graphiques que l'on peut manipuler avec le BASIC. Nous verrons un peu plus loin cet aspect des plus intéressant.

1.1. LES INSTRUCTIONS BASIC LES PLUS INTERESSANTES.

Voici maintenant une liste des instrutions les plus intéressantes (et si possibles inconnues d'autres versions du BASIC) avec leur syntaxe. La description de toutes les instructions dépasserait le cadre de cet ouvrage. Il y en a beaucoup trop !

FOLLOW, UNFOLLOW

L'instruction FOLLOW facilite la recherche des erreurs dans bien des cas. Avec cette instruction, on peut connaître le contenu de variables simples durant le fonctionnement du programme. Mais l'instruction FOLLOW ne fonctionne pas avec les variables indicées. l'affichage des variables est arrêté avec l'instruction **UNFOLLOW**.

```
FOLLOW a,angle%,texte$
UNFOLLOW texte$
```

BREAK, UNBREAK

Les instructions **BREAK** et **UNBREAK** permettent elles aussi une recherche confortable des erreurs dans un programme. Alors qu'avec d'autres BASICs il faut écrire l'instruction BREAK ou STOP dans le programme, avec le ST, il suffit d'entrer 'BREAK numéro de ligne'. Alors, après sa mise en route, le programme s'arrête automatiquement à la ligne donnée et l'affiche.Il suffit de taper la touche RETURN ou ENTER pour continuer le programme. Mais le mode BREAK n'est pas annulé. Dès que la ligne considérée est de nouveau atteinte (dans un boucle p.ex.), le programme est arrêté. Ce mode est inhibé avec l'instruction UNBREAK.

```
BREAK 120,512,2013
UNBREAK
```

TRON, TROFF

L'instruction **TRON** peut être activée uniquement en mode direct. TRON signifie **'TRACE ON'**, et **TROFF**, **'TRACE OFF'**. Si on entre TRON sans numéro de ligne, chaque numéro de ligne exécutée est affiché durant le déroulement du programme.
Avec TRON <numéro de ligne <,numéro de ligne>>, seules les lignes dont le numéro est précisé sont affichées. TROFF inhibe TRON, soit entièrement, soit uniquement pour certaines lignes.

```
TRON 50,100,121
TROFF
TROFF 121,30
```

TRACE, UNTRACE

Contrairement à TRON, **TRACE** permet l'affichage non seulement du numéro de ligne mais de toute la ligne. La syntaxe reste la même que pour l'instruction décrite précédemment.

BLOAD

Contrairement à l'instruction **LOAD** ou **OLD** simple, **BLOAD** permet de charger des secteurs complets de mémoire. On peut ainsi charger des petits programmes en langage machine ou des pages d'écran. Il faut donner comme paramètre l'adresse de chargement après le nom du fichier à charger. Cette adresse ne teste pas le secteur indiqué. Vous pourrez donc choisir n'importe quel secteur de la mémoire. Le résultat est fatal si vous ne donnez pas d'adresse de chargement. Dans ce cas, le fichier est chargé au début de la mémoire, c'est à dire à l'adresse 0 et il écrase les vecteurs d'exception. Rien n'est détruit pour autant, mais vous devrez tout recharger.

```
BLOAD "screen.bin",&H78000
```

BSAVE

Avec cette instruction, vous pourrez sauvegarder des secteurs de mémoire sur disquette. En pratique, on peut sauvegarder l'écran ou une partie de celui-ci. Mais on peut aussi sauvegarder des programmes en langage machine ou des données avec **BSAVE**. Les paramètres sont le nom du fichier, l'adresse de début et le nombre d'octets à sauvegarder.

```
BSAVE "SCREEN.bin",&H78000,&H7D00
BSAVE 'MPROG.bin',&H7FD00,768
```

Dans le premier exemple, on sauvegarde le contenu de l'écran du 520 et du 260. Pour le 520ST+, la mémoire écran commence en &HF8000.

Après la mémoire écran se trouve un secteur de 768 octets qui demeure inutilisé dans le ST. On peut y installer de petits programmes en langage machine qui ne seront pas dérangés par le BASIC ou par le système d'exploitation. Ce secteur est sauvegardé sur disquette dans le second exemple. L'adresse est celle d'un ST avec 512 Koctets. Le BASIC ATARI permet d'appeler des programmes en langage machine de deux manières. Il existe tout d'abord l'instruction **CALL**. Cette instruction est suivie de l'adresse de la routine désirée et éventuellement de valeurs nécessaires à la routine. De toute manière, l'adresse de la routine doit se trouver dans une variable.

```
adresse=&H7FD00
CALL adresse

adresse=&H7FD00
valeur1=33.33
valeur2%=100
Z$='test'
CALL adresse(valeur1,valeur2%,100,z$,"chainevide")
```

Dans le premier exemple, la routine est appelée sans paramètre. La routine peut modifier tous les registres. Masi le 'stackpointer' (pointeur de pile) doit être exact à la fin de la routine, car elle doit se terminer avec RST. Si le pointeur de pile ne convient pas, une mauvaise adresse retour est employée. Le second exemple montre comment préciser les paramètres dans l'instruction CALL. Les paramètres doivent se trouver entre parenthèses et être séparés par des virgules.
Tous les types de variables sont possibles. Mais les paramètres sont transformés en nombres signés sur 32 bits sans partie décimale. Donc notre variable 'valeur1' a permis d'envoyer une valeur de 33 à la routine.

Pour les chaînes de caractères (Z$, 'chaînevide'), l'adresse de la chaîne est envoyée, c'est à dire un nombre sur 32 bits. Mais comment saisir les valeurs dans le programme en langage machine?
Lors du branchement à la routine en langage machine, les registres **A0, A7** et **D0** sont très importants. A0 contient l'adresse de la routine. Vous pensez sans doute que c'est inutile, l'adresse d'une routine étant normalement connue. Voila de quoi vous surprendre. Nous vous expliquerons plus loin en quoi le contenu de A0 peut être utile.
Le registre D0 contient dans le mot de poids faible (les 16 bits inférieurs) le nombre de valeurs indiquées. Cette valeur également est très utiles pour certaines applications, en particulier lorsque la routine peut être apellée avec des nombres différents de paramètres. l'utilisation d'un compteur 16 bits est presque inutile. On n'utilisera jamais autant de paramètres dans une ligne BASIC.
A7 enfin, le USER-STACKPOINTER, contient l'adresse de retour dans l'interpréteur BASIC. Mais il n'y a pas seulement l'adresse de retour sur la pile. Avec l'instruction MOVE.W 4(SP),Dx, le registre Dx prend pour valeur le nombre de paramètres. Avec l'instruction MOVE.L 6(SP),Ax, on atteint le début d'une table. Cette table contient autant de mots longs (valeurs sur 32 bits) que de paramètres dans l'instruction CALL. Ces mots longs contiennent les valeurs ou les adresses des chaînes dans le format décrit.

PEEK

Bien que la fonction **PEEK** soit connue sur les autres BASICs, elle dispose de certaines particularités sur le ST. L'instruction donne des valeurs sur 16 bits. Avec PEEK(0), le contenu de la case mémoire 0 est lu en octet de poids faible et celui de la case 1 en octet de poids fort. Ce mécanisme est supprimé avec l'instruction **DEF SEG**. L'instruction PEEK ne donne alors que des valeurs sur 8 bits (voir DEF SEG). Seconde particularité : PEEK peut également donner des valeurs sur 32 bits avec la fonction **VARPTR**. Cette possibilité ravira ceux qui adorent les trucs avec les variables. Un de ces trucs est l'inscription de programmes en langage machine dans les variables. Avec la ligne :

```
adresse=PEEK (VARPTR (A$)+2)
CALL adresse
```

l'adresse sur 32 bits de la chaîne A$ est chargée dans la variable 'adresse' et le programme en langage machine contenu dans la chaîne de caractères est exécuté. Nous verrons d'autres trucs de ce genre plus tard.

POKE

l'instruction **POKE** se comporte comme l'instruction PEEK. On peut donc modifier avec POKE des valeurs sur 1,2 ou 4 octets. Après l'instruction **DEF SEG**, il faut utiliser une valeur sur un octet. Pour des valeurs plus grandes, seul l'octet de poids faible sera pris en considération. Les valeurs sur 4 octets sont disponibles avec la fonction **VARPTR**.

DEF SEG

l'instruction **DEF SEG** établit l'adresse d'un segment pour les instructions PEEK et POKE. DEF SEG ou **DEF SEG=0** fixe le segment à l'adresse physique 0. c'est l'état disponible à l'allumage. Si on donne une valeur supérieure à 0 comme paramètre, le segment pour PEEK et POKE sera à cette adresse. Voici un exemple pour éclaircir ce point. Pour pouvoir modifier et lire la RAM située après l'écran, on peut utiliser l'adresse absolue, c'est à dire l'adresse physique.

```
valeur=PEEK(&H7FD00)
```

les adresses données avec PEEK et POKE sont relatives par rapport au début du segment.

```
DEF SEG=&H7FD00
valeur=PEEK(0)
```

Mais faites attention au fait que vous obteniez dans le premier exemple le contenu des adresses &H7FD00 et &H7FD01 et dans le deuxième exemple, seulement le contenu de l'adresse &H7FD00.

DEF USR, USR, USR0 - USR9

Ce groupe d'instructions constitue la seconde manière d'appeler les programmes en langage machine à partir du BASIC. Malheureusement, ce groupe n'est pas encore entièrement développé. Lors de la définition de l'adresse de branchement, l'interpréteur indique dans la pré-version qu'une partie de programme dont le nom est B.EVST n'est pas encore mise en oeuvre. Un nouvel appel de la fonction USR conduit au message laconique 'function not yet done'. Il faudra donc attendre...à moins que vous possédiez une version définitive.

GOTOXY pos.x, pos.y

L'instruction **GOTOXY** sert à positionner le curseur à l'écran. Une instruction d'affichage (**PRINT** ou **WRITE**) commencera donc à l'emplacement indiqué. La position du curseur indiquée avec cette instruction est également utilisable pour l'instruction **INPUT**.
Les paramètres précisent les coordonnées X et Y où il faut mettre le curseur. On peut utiliser aussi bien des variables que des constantes.
L'instruction **GOTOXY** doit malheureusement encore être améliorée car la position X n'est pas correctement adaptée. La valeur donnée est multipliée par deux, ce qui est incorrect. On obtient donc des résultats abérrants. Si après l'affichage, on 'clique' sur un icône de scrolling, l'affichage est réinitialisé et on obtient les positions correctes. Vous devrez donc utiliser cette instruction avec précautions pour l'instant. La création de masque d'écran n'est pas encore très pratique.

```
GOTOXY 10,10: PRINT 'ici se trouve l'emplacement 10,10'
GOTOXY pos.x, pos.y: INPUT valeur
```

INKEY$

Si une 'fonction' doit 'fonctionner', alors la 'fonction' **INKEY$** porte mal son nom ! Elle est aussi truffée d'erreurs. Tout simplement, aucun caractère n'est lu au clavier. La raison est certainement qu'avant d'exécuter cette fonction, on teste si les touches <control-G> ou <control-C) sont pressées. Dans ce cas, le programme est soit arrêté, soit poursuivi.
Mais lors de ce test, le contenu du tampon de clavier est vidé. La fonction INKEY$ est tellement rapide qu'aucune touche pressée n'est chargée dans le tampon durant son exécution. Ainsi la fonction ne donne aucune valeur de touche. Pour beaucoup d'applications cependant, il est possible de remplacer INKEY$ par les fonctions **INPUT$** et **INP**. Nous allons immédiatement en parler.

INPUT$

La fonction INPUT$ est relativement inhabituelle et rarement présente dans les différentes versions du BASIC (p.ex. IBM PC). Avec cette fonction, on peut saisir un ou plusieurs caractères au clavier ou dans un fichier. l'intérêt de cette fonction est qu'aucune (ou presque aucune) interprétation des codes de contrôle n'est effectuée. La syntaxe est :

```
text$=INPUT$ (10)
a$(i)=INPUT$ (10,1)
```

ou

```
a$(i)=INPUT$ (10,#1)
```

Dans le premier cas, 10 caractères sont lus au clavier, sans que ces caractères ne soient affichés à l'écran. On peut presser les touches<ENTER>,<RETURN>,<Contrl-G>et<Contrl-C>sansarrêterla saisie. Le seul moyen de l'arrêter, hormis si l'on a atteint le nombre de caractères spécifié, est de presser <Contrl-Z>.

Ce caractère dont la valeur ASCII est 26 est considéré dans un fichier comme le caractère de fin de fichier. La saisie de 10 caractères est sans doute rarement utile (p.ex. saisie d'un mot clé sans affichage). Si l'on réduit le nombre de caractères à saisir à 1, cette fonction remplace la construction suivante :

```
10 a$=INKEY$: IF a$="" THEN 10
```

Les touches spéciales de l'ATARI, c'est à dire les touches de fonction et celles du bloc curseur, n'ont aucune valeur ASCII. Ces touches ne peuvent être saisies avec la fonction INPUT$.
Dans le premier et le deuxième exemple, 10 caractères sont lus dans un fichier déjà créé et sont chargés dans une variable. Lorsque l'on travaille avec des données de longueur fixe, on peut charger sans problème des caractères comme les virgules, les points virgules, deux points, et CR (touche ENTER). Pour beaucoup d'applications cela s'avère très pratique si le fichier peut être lu caractère par caractère. C'est possible en particulier avec pour paramètre de la fonction INPUT$, une longueur de 1.

INP, OUT

Dans les ordinateurs basés sur le Z80 ou le 8080, les fonctions **INP** et **OUT** servent souvent pour adresser les ports du microprocesseur. Mais étant donné que le 68000 n'admet pas d'adressage des ports, on peut se demander à quoi servent ces instructions dans l'ATARI et quelle est leur effet.
Dans le BIOS du ST se trouvent trois routines nommées **BCONSTAT, BCONIN** et **BCONOUT**. Ces trois routines traitent les entrées/sorties du système en ce qui concerne le clavier, l'écran, l'imprimante, l'interface V24, l'interface MIDI et le processeur de clavier. En assembleur, les routines sont ainsi conçues que l'on peut préciser le numéro de l'appareil désiré. Les numéros sont les suivants :

numéro	**appareil/interface**
0	Interface centronics/imprimante
1	Interface V24
2	Console : clavier et écran
3	Port MIDI
4	Processeur de clavier

Ces numéros peuvent être utilisés dans les instructions INP et OUT. On peut donc manipuler les interfaces directement du BASIC.
l'ordre :

```
OUT (0),65
```

par exemple, envoie la valeur 65 (code ASCII de 'A') vers l'imprimante. Vous estimez que l'ordre :

```
LPRINT 'A'
```

fonctionne aussi bien ? Dans ce cas précis, vous avez raison. Mais essayez donc d'envoyer le caractère LF dont le code ASCII est 10 avec l'instruction LPRINT.

Vous vous rendrez vite compte que vos efforts sont vains. Le BASIC du ST n'envoie pas les caractères CR et LF, c'est à dire les codes ASCII 13 et 10, à volonté. Cela est particulièrement désagréable pour imprimer des graphiques avec les imprimantes EPSON (et compatibles). Pas plus les bits que le nombre d'octets graphiques donné ne correspondent.
L'ATARI envoie les codes CR/LF tous les 72 caractères, ce qui imprime le graphisme n'importe comment. Les programmeurs du BASIC auraient dû tirer les leçons des erreurs d'autres programmeurs.
Mais pourquoi se plaindre ? L'instruction OUT nous sort de ce pétrin.
L'instruction OUT permet d'adresser autre chose que l'imprimante. Les autres interfaces peuvent être elles aussi programmées. On peut se servir de l'interface V24 en BASIC.

C'est particulièrement intéressant pour cette interface car on peut également lire des caractères sur la V24 avec l'instruction INP. On peut donc écrire sans difficultés un programme de terminal sur ST en BASIC.
Les instructions INP et OUT avec le numéro de périphérique 2, permettent de se servir de GEM. Tout l'écran est alors disponible. Les fenêtres et autres icônes ne sont plus utilisés. Pour la saisie, l'instruction INP (2) se comporte comme l'instruction INPUT$. Mais INP(2) possède un avantage décisif. Les touches de fonction ainsi que celles du bloc curseur ont des valeurs spécifiques. Avec le numéro 4, l'interface MIDI peut être programmée aussi bien en entrée qu'en sortie. Les utilisateurs qui possèdent des instruments correspondant, c'est à dire les orgues et les synthétiseurs, peuvent les diriger à partir du ST. Il faut naturellement qu'ils soient munis d'un port MIDI. Avec ces informations, il est relativement facile de diriger des instruments à partir du BASIC.
Le dernier périphérique qu'on peut manipuler est le clavier. Comme vous le savez sans doute, l'ATARI dispose d'un clavier 'intelligent'. Il contient un processeur qui gère les touches, la souris et le joystick. De plus, le processeur de clavier possède une horloge. Cependant, on ne peut qu'envoyer des valeurs sur le processeur de clavier, sans grand effet. La réponse est rendue par le système d'exploitation du ST. Un INP(4) donne toujours la valeur 16.

VARPTR

VARPTR est une fonction dont le résultat est une adresse.
L'argument de la fonction VARPTR est soit une variable, soit un numéro de fichier (!):

```
a$="test"
ad=VARPTR (a$)
a=10
adr=VARPTR (a)
open "I",1,"dudu.dat"
? VARPTR (#1)
```

Les valeurs obtenues en retour du dernier appel peuvent être interprétées de différentes manières. Vous découvrirez peut être seul leur signification. Mais ce point ne sera bien éclairci qu'avec un manuel du BASIC.
Dans les deux premiers cas, l'interprétation est simple. Voyons d'abord le premier cas.
Après l'appel de VARPTR, la variable contient l'adresse du descripteur de type de la variable a$. Le descripteur de type est sur 6 octets. Le premier octet contient un flag dont nous allons bientôt expliquer le rôle. Le deuxième octet du descripteur donne la longueur de la chaîne. Une chaîne peut donc faire au plus 255 caractères.
Dans les troisième à sixième octets se trouve l'adresse mémoire à laquelle est la chaîne.
Si vous essayez ces valeurs avec l'exemple donné précédemment, vous serez certainement surpris. l'adresse est en fait la chaîne elle même. Toutes les chaînes qui font 1 à 4 caractères sont directement inscrites dans l'adresse du descripteur de type.
La signification du flag est donc évidente. Si il est nul, la chaîne fait moins de 5 caractères et elle est chargée directement dans le descripteur de type. Si la valeur hexadécimale du flag est 10, alors les octets 3 à 6 contiennent l'adresse de la chaîne.
La fonction VARPTR donne pour les variables numériques l'adresse mémoire à laquelle se trouve le nombre. Les nombres réels sont sauvegardés sur 4 octets, les INTEGER (ex.:A%) sur 2 octets. Nous analyserons les vecteurs INTEGER plus particulièrement car ils constituent une bonne mémoire protégée pour les programmes en langage machine.

SOUND

L'instruction **SOUND** du BASIC ST est puissante et facile à programmer.
Le processeur sonore du ST est un circuit intégré YM-2149. Il est compatible avec le circuit connu AY-3-8910, qui est disponible sur de nombreux ordinateurs (CPC, MSX..). Ce circuit offre de nombreuses possibilités pour créer des sons sur trois canaux.

De plus, on peut produire du bruit blanc, qui permet de faire des son spéciaux (p.ex.: batterie).
L'instruction SOUND a au plus 5 paramètres qui sont des valeurs numériques inscrites à la suite de l'instruction. Une instruction complète est donc:

SOUND canal,volume,fréquence,octave,durée

La valeur du canal varie de 1 à 3 suivant le canal désiré, La valeur du volume varie de 1 à 15, 1 étant faible et 15 indiquant la pleine puissance. Cette valeur est mise suivant le canal dans les bits 0 à 3 du registre 8 (canal A), 9 (canal B), ou 10 (canal C). La variable 'fréquence' varie de 1 à 12. Etant donné qu'une octave est composée de 12 tons, on peut jouer directement avec des notes. L'octave peut varier de 1 à 8. Le ST peut donc jouer des sons sur 8 octaves. La durée peut prendre des valeurs allant de 1 à 255. La durée est fixée en 20 millisecondes. Si vous donnez une valeur de 50, la note durera environ 1 seconde.
La table qui suit vous indique comment obtenir les notes avec la variable fréquence.

1 do	**2 do#**	**3 ré**
4 ré#	**5 mi**	**6 fa**
7 fa#	**8 sol**	**9 sol#**
10 la	**11 la#**	**12 si**

Le la mineur (440 hz) est obtenu avec l'instruction :

SOUND 1,15,10,4,255

La quatrième octave correspond donc normalement à l'octave nulle. Des valeurs d'octaves inférieures donnent des sons plus graves et des valeurs plus grandes produisent des sons plus aigus.

WAVE

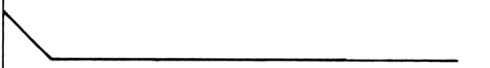
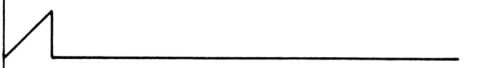
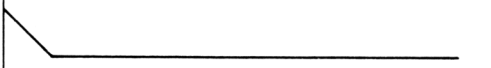
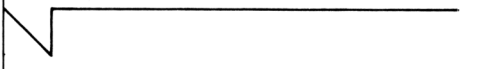
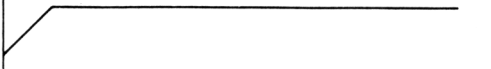
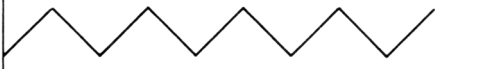
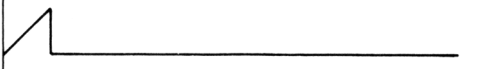
L'instruction SOUND permet déjà de créer de jolies mélodies sur une voix.
L'instruction **WAVE**, elle, permet la programmation polyphonique. Elle offre de nombreuses possibilités. Mais tout n'est pas très simple. Nous avons eu beaucoup de difficultés à comprendre la structure et les paramètres. De plus, pour bien la maîtriser, il est utile de connaître la structure du circuit sonore.
Comme l'instruction SOUND, WAVE comprend cinq paramètres. Le premier peut être comparé au paramètre 'canal' de SOUND. Il permet de choisir un canal pour produire un son. Mais les valeurs sont quelque peu différentes.
On comprendra mieux les valeurs du paramètre en analysant un registre particulier du chip sonore. Il s'agit du registre 7 qui est un registre multi-fonctions. Chaque bit du registre à une signification particulière.

Les bits 0 à 2 permettent d'ouvrir ou de fermer les trois canaux sonores. Si le bit 0 du registre 7 est positionné, le canal A est fermé Le bit 1 sert pour le canal B et le bit 2, pour le canal C. Un bit mis à zéro ouvre le canal.
Les bits 3 à 5 servent à activer la fonction de bruit blanc pour les trois canaux. De même, la fonction est activée avec un bit positionné et elle est inhibée avec un bit à 0.
Les bits 6 et 7 permettent de programmer les ports 8 bits intégrés dans le chip sonore. Ces deux bits ne sont d'aucune importance pour la création de sons, aussi nous ne nous y attarderons pas.
Ce sont les bits 0 à 5 qui peuvent être manipulés avec le premier paramètre de l'instruction WAVE. Si nous considérons le paramètre comme un nombre binaire, les bits correspondants du paramètre ont la fonction opposée. Si le paramètre prend pour valeur 1, le bit 0 du registre 7 est effacé et le canal A est activé. Tous les autres bits du registre 7 sont positionnés et les fonctions correspondantes sont désactivées. Si le premier paramètre est p.ex. 37 (en binaire %100101), alors les canaux A et C sont activés et le canal B est fermé.

De plus, la source de bruit blanc est activée pour le canal C. Si le paramètre est nul, tous les canaux et toutes les sources de bruit blanc sont inhibés.
Le second paramètre de l'instruction WAVE influe sur les trois registres du circuit sonore. Il s'agit des registres 8, 9 et 10. Cependant, tous les bits de ces registres ne sont pas modifiés, mais seulement le bit 4. Ce bit 4 des trois registres indique si le volume des trois canaux sera fixé par l'instruction SOUND (bits 0 à 3 des trois registres : voir l'instruction SOUND) ou bien par une enveloppe réalisée au niveau du hardware.
Cette enveloppe est une des particularités du chip sonore. Avec cette enveloppe, le volume du son est modifié périodiquement ou bien une fois pour toutes. Cette enveloppe permet de faire varier la tonalité d'un son de multiples façons.
Le second paramètre lui aussi doit être considéré comme un nombre binaire. La signification des bits est la suivante:

Bit n°	**fonction**
1 positionné	volume du canal A avec les bits 0-3 du registre 8
1 effacé	volume du canal A avec l'enveloppe
2 positionné	volume du canal B avec les bits 0-3 du registre 9
2 effacé	volume du canal B avec l'enveloppe
3 positionné	volumee du canal C avec les bits 0-3 du registre 10
3 effacé	volume du canal C avec l'enveloppe

Ainsi la valeur peut varier de 0 à 7. Si le second paramètre est 0, le volume de chacun des canaux est déterminé avec l'instruction SOUND. Avec pour valeur 5, le volume des canaux A et C est fixé par l'enveloppe et celui du canal B est déterminé par les bits 0 à 3 du registre 9.
Le troisième paramètre de l'instruction WAVE est en relation étroite avec le second paramètre. Ce paramètre choisi une des 9 enveloppes possibles.

REG. 15				
B3	B2	B1	B0	
CONTINUE	ATTACK	ALTERNATE	HOLD	
0	0	—	—	
0	1	—	—	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

Sa valeur peut varier de 0 à 15, mais certaines valeurs donnent des enveloppes identiques. Les résultats ne sont pas descriptibles. Aussi, nous vous donnons les enveloppes correspondant aux différentes valeurs dans le schéma ci- joint.
Le quatrième paramètre également manipule directement les registres du chip sonore. Ce circuit dispose en effet de deux registres 8 bits qui déterminent la période de l'enveloppe. On comprend donc que le quatrième paramètre puisse prendre des valeurs variant de 0 à 65535. Plus le paramètre est grand, plus la période est grande. Pour de très petites valeurs (<1000), la fréquence est tellement élevée qu'on dispose également d'une fréquence audible en plus. Elle est disponible pour produire certains effets spéciaux.

Le cinquième paramètre fixe la longueur du son comme pour l'instruction SOUND.
Il n'est effectif dans un programme que si une autre instruction SOUND ou WAVE se trouve à la suite. En mode direct ou sous l'éditeur, le son est produit jusqu'à ce qu'une touche soit pressée, c'est à dire jusqu'au déclic de la touche.

LINEF

L'instruction **LINEF** est la fonction graphique la plus simple du BASIC ST. Avec cette instruction, on peut tracer des lignes ou des points à l'écran. On utilise pour cela quatre paramètres afin de préciser les points de début et de fin de la ligne. Pour dessiner un point, il suffit de donner les mêmes coordonnées pour le point de début et de fin. Nous apprendrons plus loin comment changer non seulement l'épaisseur du trait, mais aussi le type de la ligne et l'apparence du point de début et du point de fin.

```
LINEF 10,10,50,40
```

CIRCLE

L'instruction **CIRCLE** permet de dessiner des cercles ou des arcs de cercle.

Trois ou cinq paramètres sont nécessaires pour cela. Pour dessiner un cercle, trois paramètres suffisent. Les deux premiers paramètres précisent les coordonnées du centre du cercle et le troisième indique le rayon du cercle. Les quatrième et cinquième paramètres ne sont utiles que pour dessiner un arc de cercle. Il permettent de préciser les angles de départ et d'arrivée en degrés.
Les paramètres sont donnés en 1/10 ème de degré. Pour dessiner un demi cercle d'un rayon de 100 points on procédera ainsi :

```
CIRCLE 320,199,100,0,1800
```

Avec cette instruction vous verrez où se trouve le point 0 pour indiquer l'angle. Le cercle est toujours dessiné dans le sens inverse des aiguilles d'une montre, à partir de la position '3 heures' sur une montre.
L'instruction CIRCLE elle aussi utilise des traits dont l'épaisseur fait un pixel. Mais nous verrons comment modifier cela.

La dernière particularité notable est que le cercle n'est pas un cercle en réalité. l'instruction CIRCLE dessine un polygone dont la forme s'approche du cercle. Avec un rayon de 30, on voit clairement qu'il ne s'agit pas d'un cercle, mais d'un octogone. Si vous désirez dessiner un vrai cercle, il faudra calculer les valeurs et utiliser l'instruction LINEF.

PCIRCLE

L'instruction **PCIRCLE** elle aussi trace un simili-cercle. Les paramètres sont les mêmes que pour l'instruction CIRCLE. Mais ce cercle est rempli avec une couleur ou un motif. On précise la couleur ou le motif avec l'instruction **COLOR**.

ELLIPSE

On peut outre les cercles tracer des ellipses ou des arcs d'ellipse. Cette instruction a quatre ou six paramètres.

Les deux premiers paramètres précisent les coordonnées du centre et les deux paramètres suivants donnent les rayons de l'ellipse en abscisse et en ordonnée. Les paramètres 5 et 6 (optionnels) donnent les angles pour dessiner un arc d'ellipse. Le principe est le même que pour l'instruction CIRCLE.

```
ELLIPSE 320,200,100,30,450,2700
```

PELLIPSE

Le 'P' précédant le nom indique que l'ellipse sera remplie avec la couleur ou le motif courant. Les paramètres sont les mêmes que pour l'instruction ELLIPSE.

COLOR

L'instruction **COLOR** permet de changer la couleur d'écriture, la couleur du motif de remplissage, la couleur des lignes dessinées avec LINEF et le motif de remplissage. 5 paramètres sont nécessaires.

Le premier paramètre précise la couleur du texte à afficher. Seules des valeurs allant de 0 à 1 sont possibles avec un moniteur monochrome. Un 0 correspond à une écriture invisible, c'est à dire de la couleur du fond. Avec un moniteur couleur, la couleur dépend de la résolution choisie. Avec une basse résolution (320*200), on peut choisir entre 16 couleurs (0 à 15), en mode moyenne résolution, on peut utiliser une valeur allant de 0 à 3.

Le second paramètre donne la couleur de l'instruction PCIRCLE, PELLIPSE et **FILL** . Les valeurs correspondent à celles du premier paramètre.

Le troisième paramètre précise lui aussi une couleur, mais pour les instructions graphiques CIRCLE, ELLIPSE, et LINEF. Comme on le voit, on peut disposer de beaucoup de couleurs avec un moniteur couleurs.

Les deux derniers paramètres indiquent le motif de remplissage pour les instructions FILL, PCIRCLE et PELLIPSE. Le cinquième paramètre peut varier de 0 à 4 et permet le choix de base du motif.
- Si cette valeur est 0, aucun motif n'est dessiné, quels que soient les autres paramètres.
- Si la valeur est 1, aucun motif n'est dessiné, mais la figure est remplie avec la couleur des caractères.
- Avec pour valeur 2, le motif est dessiné avec la couleur des caractères. La forme du motif (il y en a 24 différents) est fixée avec le quatrième paramètre.
- Si le cinquième paramètre est 3, on peut choisir d'autres motifs. Ces paramètres dessinent diverses hachures.
- Si le cinquième paramètre a la valeur 4, on choisi un motif définissable sous GEM. Les développeurs d'ATARI on dessiné le symbole ATARI comme motif définissable.

FILL

L'instruction **FILL** permet de remplir des surfaces. Les valeurs fixées par l'instruction COLOR sont utilisées. Les paramètres sont les coordonnées d'un point situé à l'intérieur de la surface à remplir.
En outre, on peut fixer un troisième paramètre. Dans ce cas, cette valeur correspond à la couleur à laquelle doit s'arrêter la routine de remplissage.
Sans ce paramètre, la routine s'arrête à tous les points d'une couleur différente de celle du fond.

FULLW

Cette instruction est la première d'une série d'instructions qui permettent de manipuler les fenêtres. Avec l'instruction **FULLW**, vous pouvez donner la taille maximale à chacune des quatre fenêtres. Les quatre fenêtres disponibles en BASIC correspondent aux valeurs suivantes :

0 EDIT-window
1 LIST-window
2 OUTPUT-window
3 COMMAND-window

Avec la ligne :

FULLW 2

La fenêtre d'affichage prend la taille de l'écran. Les autres fenêtres sont recouvertes.

CLEARW

Cette instruction sert à effacer le contenu d'une fenêtre. Elle correspond à l'instruction CLS du BASIC MICROSOFT. Mais elle est liée à une fenêtre spécifique en BASIC ST. Après la ligne

CLEARW 2

La fenêtre d'affichage est effacée. La position du curseur n'est pas modifiée par cette instruction. Vous devrez donc souvent faire suivre l'instruction CLEARW 2 d'un GOTOXY 0,0. Ainsi le curseur se retrouvera dans le coin gauche en haut de l'écran.

CLOSEW

Avec l'instruction **CLOSEW,** on peut effacer les fenêtres. Elles disparaissent alors totalement de l'écran. Les numéros des fenêtres sont les mêmes que pour les autres instructions de manipulation des fenêtres.

OPENW

Avec l'instruction **OPENW,** on peut ouvrir de nouveau des fenêtres fermées (CLOSEW). Mais cette instruction ne fonctionne que si au moins une fenêtre est ouverte.

Cela semble être une nouvelle erreur du BASIC.

RESUME

Nous voici arrivés à la fin de la description des instructions spécifiques du BASIC ST. Toutes les autres instructions ou fonctions du BASIC fonctionnent comme dans les autres versions de BASIC. Il existe beaucoup de livre les décrivant ; aussi ne nous attarderons nous pas dessus.

Nous avons volontairement ignoré deux instructions que vous ne rencontrerez sans doute pas dans d'autres BASIC. Il s'agit de **GEMSYS** et de **VDISYS**. Etant donné qu'elles sont très puissantes, nous leur consacrerons un chapitre en particulier. Vous serez étonné par les possibilités qu'elles ajoutent au BASIC.

1.2. BASIC ET GEM

Outre les possibilités incluses dans le BASIC, il en existe de nombreuses autres, en particulier avec les fonctions disponibles sous GEM. nous allons décrire ces fonctions dans ce chapitre. Voyons d'abord ce qu'est GEM et quelques unes de ses applications les plus courantes. GEM, abréviation de GRAPHIC ENVIRONMENT MANAGER, est un système qui a été développé sur des systèmes informatiques graphiques, afin de permettre la création de programmes graphiques indépendamment de l'appareil. GEM permet également d'envoyer les résultats non seulement à l'écran mais aussi sur une imprimante matricielle, une table graphique ou tout autre périphérique graphique.

Vous vous demandez sans doute comment un programme peut fonctionner sur plusieurs appareils. Voyons tout d'abord comment est constitué l'environnement CP/M. Un programme, comme WORDSTAR ou M-BASIC ne fonctionnera sur différents ordinateurs munis du système d'exploitation CP/M qu'à condition de répondre à des spécifications techniques très précises et d'être indépendants de l'appareil sur lesquels ils ont été développés. Il en va de même pour GEM. Si un programme s'en tient aux possibilités de GEM, il fonctionnera sans modification sur les ordinateurs munis de GEM. Naturellement, un programme écrit pour le 68000, c'est à dire pour le micro processeur de l'ATARI ne fonctionnera pas sur un IBM-PC sous GEM. Mais si les programmes sont écrits dans un langage évolué (p.ex.: C, Pascal, Modula2), il suffira d'exécuter une compilation sur le nouveau matériel pour que le programme fonctionne. En pratique, il faudra sans doute certaines adaptations car les programmes utilisent la plupart du temps quelques spécificités de la machine sur laquelle ils ont été écrits. Mais si l'on tient scrupuleusement compte de ces faits, le programme pourra presque être compilé sans modification.

GEM est divisé en deux modules : L'AES et le VDI. L'AES est la partie qui gère les fenêtres, le menu, les icônes et d'autres types d'objets.

L'AES contient donc les fonctions les plus complexes décrites plus haut. La plupart des fonctions de l'AES n'ont pas été prévues pour être appelées à partir du BASIC.

Mais il existe des exceptions.
Le VDI contient les routines graphiques pour l'affichage du texte, du graphisme et pour la saisie de texte ou le test du clavier ou de la position de la souris. Le VDI est divisé en GDOS (graphic device operating system) et DEVICE DRIVER. Le device driver est la partie de GEM dépendant du matériel et doit être adaptée en fonction des périphériques de sortie. Dans la version actuelle de GEM pour le ST, le seul device driver disponible est l'interface avec le moniteur. (device driver = gestionnaire de périphériques) D'autres driver seront sans doutes développés par la suite.

1.2.1 L'instruction VDISYS

L'instruction VDISYS constitue l'interface avec le VDI. Si cette instruction est exécutée dans un programme, une des multiples fonctions du VDI est appelée et exécutée. Pour pouvoir appeler une fonction, il faut donner au VDI certains paramètres et le numéro de la fonction.
Le VDI utilise cinq vecteurs (ou adresses mémoires) pour permettre de préciser les paramètres. Les vecteurs ont pour noms: **CONTRL, INTIN, INTOUT, PTSIN** et **PTSOUT**. Les programmeurs qui ont développé le BASIC ST ont trouvé les possibilités du VDI tellement intéressantes qu'ils ont réservé des variables dans le BASIC avec pour noms ceux des vecteurs. Vous obtiendrez les adresses des vecteurs avec :

```
? contrl; intin; intout; ptsin; ptsout
```

remarque : les vecteurs susnommés ne sont pas les vecteurs adressés par le VDI. Mais les valeurs des vecteurs du BASIC sont automatiquement chargées dans les vecteurs du VDI.

Comme on dispose des vecteurs, il n'y a plus rien à ajouter pour appeler le VDI. Un exemple vous donnera la marche à suivre.

```
POKE CONTRL, (numéro de l'appel)
POKE CONTRL+2, (nombre de paramètres dans INTIN)
REM CONTRL+4, (nombre de paramètres dans INTOUT)
POKE CONTRL+6, (nombre de paramètres dans PTSIN)
REM CONTRL+8, (nombre de paramètres dans PTSOUT)
POKE CONTRL+10,(No secondaire éventuel de la fonction)
POKE CONTRL+12, (signal périphérique, entre 1 et 10)
REM
POKE INTIN, (premier paramètre)
POKE INTIN+2, (deuxième paramètre)
POKE INTIN+4, (troisième paramètre)
:
jusqu'à
:
POKE INTIN+n, (dernier paramètre)
REM
POKE PTSIN, (premier paramètre)
POKE PTSIN+2, (deuxième paramètre)
POKE PTSIN+4, (troisième paramètre)
:
jusqu'à
:
POKE PTSIN+n, (dernier paramètre)
REM
VDISYS
REM
```

Dans cet exemple, vous voyez comment mettre les paramètres dans les vecteurs correspondants, à l'aide de POKE. Comme les paramètres tiennent tous sur 16 bits, une instruction POKE suffit pour les charger. Vous comprendrez ainsi pourquoi les vecteurs CONTRL, CONTRL+4 et CONTRL+8 ne sont pas modifiés par POKE. Après un appel, on trouve dans ces vecteurs le nombre de paramètres retourné par la fonction dans INTOUT et PTSOUT.
Un exemple pratique va rendre cela plus clair. Dans cet exemple, un appel du VDI nous permettra d'allumer le curseur de la souris qui normalement n'est pas visible.

Pour cela, nous avons besoin de connaître le numéro de la fonction et les paramètres nécessaires. Ces informations se trouvent dans la documentation sur GEM fournie par DIGITAL RESEARCH, la firme qui a développé GEM. Mais vous trouverez également ces données dans 'le livre du GEM' paru chez MICRO APPLICATION. Nous décrirons ici uniquement quelques fonctions particulièrement utiles, afin de rester dans le cadre de ce livre.

Le CODEOP nécessaire, c'est à dire le numéro de la fonction est 122. La fonction porte le nom de **SHOW MOUSE**. Il faut porter le numéro de la fonction, donc 122, dans CONTRL. Le vecteur PTSIN ne donnera aucune valeur en retour de la fonction SHOW MOUSE, aussi CONTRL+2 doit être mis à 0. Le vecteur INTIN nous donnera un paramètre en retour. Aussi CONTRL+6 devra prendre la valeur 1.

Aucune valeur n'est attendue dans CONTRL+10. Cet élément contient certaines spécifications pour quelques fonctions du VDI. CONTRL+12 contient le gestionnaire de périphériques (**DEVICE HANDLE**). A la mise en route de BASIC, il prend la valeur 2. En effet, étant donné que l'affichage à l'écran utilise n'importe quelle valeur entre 1 et 10, il n'est pas nécessaire de modifier la valeur. Si la valeur était comprise entre 11 et 20 l'affichage se ferait sur un traceur, et pour l'intervalle 21-30, il s'agirait d'une imprimante.
Ce sont tous les paramètres du vecteur CONTRL. Il reste à déterminer la valeur utilisée dans INTIN. d'abord une petite explication :

Outre la fonction SHOW MOUSE, il existe une fonction **HIDE MOUSE** qui éteint le curseur de la souris. Le VDI prend en compte le nombre d'appel de HIDE MOUSE.
Si SHOW MOUSE est appelé alors que INTIN à une valeur différente de 0, le nombre d'appels de HIDE MOUSE est décrémenté de 1. Le curseur n'est donc pas obligatoirement visible après l'appel de la fonction.
Si INTIN à pour valeur 0, le nombre d'appels de HIDE MOUSE est ignoré et le curseur est obligatoirement affiché.

Voici le listing de l'exemple:

```
10 POKE CONTRL, 122
20 POKE CONTRL+2, 0
30 POKE CONTRL+6, 1
40 REM
50 POKE INTIN, 0 : REM affichage du curseur obligatoire
60 REM
70 VDISYS
80 REM
```

Après l'appel, CONTRL+4 et CONTRL+8 auront pour valeur zéro. Cela indique qu'aucune valeur n'a été donnée en retour dans INTOUT et PTSOUT.

1.2.2. Routines utiles du VDI en BASIC

Presque chaque routine du VDI est utilisable en BASIC n'importe quand.
Cependant, certaines routines sont inutiles car elles correspondent à des fonctions déjà disponibles en BASIC. Ainsi, il est superflu de dessiner une ligne avec l'instruction VDISYS. L'instruction LINEF est beaucoup plus pratique. L'affichage de texte avec le VDI est lui aussi rarement utile.
Cependant, ces effets peuvent avoir parfois un certain intérêt. Testez les exemples et vous verrez si ils peuvent vous être utiles.
Nous allons d'abord montrer les effets possibles avec l'affichage du texte.

EFFETS SPECIAUX AVEC LE TEXTE
Set Graphic Text Special Effect

Cette fonction dont le numéro est 106 peut avoir de multiples utilisations.
Elle permet de modifier l'aspect des caractères sur l'écran. Voici tout de suite un exemple:

```
100 FULLW 2: CLEARW 2
110 a$    ="aspect normal          , intin="
120 a$(0)="traits gras             , intin="
130 a$(1)="contrasté               , intin="
140 a$(2)="italiques               , intin="
150 a$(3)="souligné                , intin="
160 a$(4)="encadré                 , intin="
170 GOTOXY 6,3
180 ?a$;i
190 FOR i=0 to 4
200 GOTOXY 6,5+2*i
210 POKE CONTRL, 106
220 POKE CONTRL+2, 0
230 POKE CONTRL+6, 1
240 POKE INTIN, 2^i
250 VDISYS
260 ? a$(i) ; 2^i
270 next
280 POKE CONTRL, 106
290 POKE CONTRL+2, 0
300 POKE CONTRL+6, 1
310 POKE INTIN, 0
320 VDISYS
330 a=inp(2)
```

Cet exemple montre tous les effets spéciaux de base. Mais il est possible de mixer ces effets. Ainsi avec la valeur 9 dans INTIN, vous obtenez l'effet 'caractères gras/ soulignés'. Dans les lignes 280 à 320, INTIN est réinitialisé en mode normal avec la valeur 0. Sans cette initialisation, l'effet reste présent. Seul l'éditeur permet d'effacer les effets de base et de revenir au mode normal.

MODIFICATION DE LA TAILLE DES CARACTERES

Set Character Cell Height, Points Mode

Outre la modification de la forme des caractères, on peut changer leur taille.

Six tailles sont possibles. Etant donné que la hauteur aussi bien que la largeur sont modifiées par cette fonction, on peut rencontrer des difficultés d'affichage pour les trois tailles les plus grandes. Une instruction PRINT est faite pour une largeur de 8 pixels. Mais étant donné que les caractères font plus de 8 pixels, leur bordure droite peut être tronquée. De toute façon les trois hauteurs les plus faibles ne posent pas de problèmes. Voici le programme:

```
10  FULLW 2: CLEARW 2
20  a$(0)="tout petit              , intin="
30  a$(1)="un peu plus grand       , intin="
40  a$(2)="normal                  , intin="
50  a$(3)="encore plus grand       , intin="
60  a$(4)="très gros               , intin="
70  a$(5)="géant                   , intin="
80  a(0)=1: a(1)=9: a(2)=10: a(3)=16: a(4)=18: a(5)=20
90  GOTOXY 6,3
100 FOR i=0 to 5
110 GOTOXY 6,5+2*i
120 POKE CONTRL, 107
130 POKE CONTRL+2, 0
140 POKE CONTRL+6, 1
150 POKE intin, a(i)
160 VDISYS
170 ?a$(i) ; a(i)
180 NEXT
190 POKE CONTRL, 107
200 POKE CONTRL+2, 0
210 POKE CONTRL+6, 1
220 POKE intin, 10
230 VDISYS
240 a = INP(2)
```

Le problème des gros caractères peut être résolu. Vous en saurez plus dans le programme qui suit.

AFFICHAGE DE TEXTE GRAPHIQUE
Text

L'affichage de texte avec la fonction VDI 8 permet d'afficher des chaînes de caractères correctement, même quand des effets spéciaux ont été utilisés. Cette fonction est particulièrement intéressante pour l'affichage de caractères agrandis avec l'instruction PRINT.

La fonction 8 utilise plusieurs paramètres. Il faut indiquer dans le vecteur INTIN la chaîne de caractères à afficher. Chaque caractère est chargé dans l'octet de poids faible d'un mot du vecteur. Dans notre exemple, le texte à afficher est chargé dans le vecteur INTIN en ligne 230. Le dernier caractère du texte doit être nul (ligne 250).

De plus, La position d'affichage doit être donnée dans le vecteur PTSIN. La position d'affichage est fonction des coordonnées de l'écran et non de données relatives à une fenêtre. Le VDI ne tient compte d'aucune fenêtre. Elles sont gérées par l'AES. Pour ces raisons, tous les affichages du VDI à l'écran peuvent se faire sur tout l'écran.
La position est donnée relativement au coin gauche en bas du caractère à afficher. Il faut éviter des valeurs qui feraient sortir les caractères de l'écran par le haut.
Voici maintenant le programme avant d'en venir à une autre particularité :

```
100 a$(0)="tout petit"
110 a$(1)="déjà plus gros"
120 a$(2)="normal"
130 a$(3)="encore plus gros"
140 a$(4)="très gros"
150 a$(5)="géant"
160 a(0)=1: a(1)=9: a(2)=10: a(3)=16:
    a(4)=18:a(5)=20
170 yp(0)=50: yp(1)=62: yp(2)=80: yp(3)=100:
    yp(4)=125: yp(5)=160
```

```
180 FULLW 2: CLEARW 2
190 for c=0 to 5
200 a=a(c):a$=a$(c)
210 GOSUB SETHIGHT
220 FOR i=1 TO LEN(a$(c))
230 POKE INTIN+(i-1)*2,asc(mid$(a$(c),i,1))
240 NEXT
250 POKE INTIN+(i-1)*2,0
260 POKE CONTRL, 8
270 POKE CONTRL+2, 1
280 POKE CONTRL+6, len(a$(c))+1
290 POKE PTSIN, 100
300 POKE PTSIN+2, yp(c)
310 VDISYS
320 NEXT c
330 a=10
340 GOSUB SETHIGHT
350 a=inp(2)
360 END
370 SETHIGHT :
380 POKE CONTRL,107
390 POKE CONTRL+2, 0
400 POKE CONTRL+6, 1
410 POKE INTIN, a
420 VDISYS
430 RETURN
```

Une autre particularité du BASIC ST se trouve dans les lignes 210 et 370. Le BASIC ST permet d'inscrire des labels dans le programme. Ces labels doivent obligatoirement se trouver en début de ligne et être terminés par un signe 'deux points'. Les deux points peuvent être suivis de n'importe quel texte.
L'avantage du label est de pouvoir être utilisé à la place d'un numéro de ligne. Les instructions GOTO, GOSUB et ON GOTO fonctionnent avec des labels, ainsi que l'instruction RESTORE qui permet de repositionner le pointeur de DATA sur des éléments spécifiques d'une ligne de DATAs.

En ligne 350, on attend la frappe d'une touche pour arrêter le programme.

MODIFICATION DU SENS D'AFFICHAGE DU TEXTE
Set Character Baseline Vector

Avec la fonction 13 du VDI, on peut modifier le sens de l'affichage d'un texte.
Pour cela, L'angle désiré doit se trouver dans INTIN. Le pas de l'angle est de 90 degrés. Pour afficher du texte avec un angle de 45 degrés, aucun algorithme de rotation n'a été prévu. La seconde caractéristique est que l'angle doit être donné en dixième de degré. 90 degrés seront donc indiqués par la valeur 900.
(ligne 320)

N'oubliez pas de remettre l'angle à zéro après chaque affichage. Sinon, la seule solution pour quitter le BASIC sera d'entrer QUIT. En effet, toutes les autres instructions sont touchées par la modification. Mais le système n'est pas remis en ordre pour autant. Le résultat est un embrouillamini indescriptible.

```
100 a$="silence, on tourne !"
110 FULLW 2: CLEARW 2
120 FOR angle=0 to 3
130 GOSUB TXT.tourne
140 FOR i=1 to len(a$)
150 POKE intin+(i-1)*2,asc(mid$(a$,i,1))
160 NEXT
170 POKE intin+(i-1)*2,0
180 POKE CONTRL, 8
190 POKE CONTRL+2, 1
200 POKE CONTRL+6,len(a$)+1
210 POKE PTSIN, 300
220 POKE PTSIN+2, 200
230 VDISYS
240 NEXT angle
250 a=inp(2)
```

```
260 angle=0: GOSUB txt.tourne
270 END
280 TXT.tourne:
290 POKE CONTRL, 13
300 POKE CONTRL+2, 0
310 POKE CONTRL+6, 1
320 POKE INTIN, angle*900
330 VDISYS
340 RETURN
```

MISE EN PLACE DES FORMATS DE LIGNES
Set Polyline Line Type

Dans la description des instructions LINEF, CIRCLE et ELLIPSE, nous avons indiqué que le format des lignes peut être modifié. Voici comment faire. Vous avez besoin de la routine VDI dont le numéro est 15. Le 520ST comprend sept types différents de ligne. Le numéro du type doit être indiqué dans INTIN. Nous n'avons pas besoin d'autres paramètres, aussi le programme de démonstration sera-t-il très court.

```
10  FULLW 2:CLEARW 2
20  i=20
30  FOR motif=1 to 7
40  GOSUB set.motif
50  FOR c=1 to 20
60  LINEF 20,c+i,500,c+i
70  NEXT c
80  i=i+30
90  NEXT motif
100 a=inp(2)
110 END
120 set.motif :
130 POKE CONTRL, 15
140 POKE CONTRL+2, 0
150 POKE CONTRL+6, 1
160 POKE INTIN, motif
```

```
170 VDISYS
180 RETURN
```

Dans ce programme, les 7 motifs sont dessinés 20 fois les uns en dessous des autres. Le septième motif apparaît comme un trait continu. Mais l'utilisateur peut le redéfinir. Nous allons le faire.

REDEFINITION DU SEPTIEME MOTIF DE LIGNE

Le numéro de fonction nécessaire pour programmer le motif est le CODEOP 113.
Dans cette fonction, INTIN prend pour valeur le motif de la ligne dessinée sur 16 bits. Il faut noter que le bit de poids fort est toujours le premier pixel, c'est à dire le pixel de gauche.

```
100 FULLW 2: CLEARW 2
110 POKE CONTRL, 113
120 POKE CONTRL+2, 0
130 POKE CONTRL+6, 1
140 POKE INTIN, &HAAAA : ' motif
150 VDISYS
160 POKE CONTRL, 15
170 POKE CONTRL+2, 0
180 POKE CONTRL+6, 1
190 POKE INTIN,7 : ' motif
200 VDISYS
210 FOR c=1 to 20
220 LINEF 20,c+i,500,c+i
230 NEXT c
240 a=inp(2)
```

A la place du motif que nous avons choisi: &HAAAA ou %1010101010101010, vous pouvez choisir ce que vous voulez. Essayez de trouver où est dessiné le bit de poids fort lorsque la ligne est dessinée verticalement.

MODIFICATION DE L'EPAISSEUR DE LA LIGNE
Set Polyline Line Width

Lorsque vous écrivez des programmes avec de nombreuses instructions graphiques, le CODEOP 16 peut vous épargner de la programmation. Il permet de modifier l'épaisseur de la ligne. Ainsi, il n'est pas nécessaire d'appeler plusieurs fois la fonction CIRCLE ou LINEF pour obtenir des lignes plus épaisses. De plus le programme va plus vite car il est plus rapide de dessiner un ligne épaisse que plusieurs minces.
il faut indiquer à la fonction 16 l'épaisseur désirée dans INTIN. Les valeurs inférieures à 3 ne doivent pas être employées pour indiquer l'épaisseur. Si vous donnez par exemple la valeur 2, la ligne aura une épaisseur d'un pixel, c'est donc la valeur normale qui est prise en compte.

```
100 FULLW 2: CLEARW 2
110 i=20
120 LINEF 20, c+i, 500, c+i
130 i=i+24
140 FOR c=3 TO 25 STEP 2
150 GOSUB set.width
160 LINEF 20, i, 500, i
170 i=i+25
180 NEXT c
190 c=2:GOSUB set.width
200 a=inp(2)
210 END
220 set.width :
230 POKE CONTRL, 16
240 POKE CONTRL+2, 1
250 POKE CONTRL+6, 0
260 POKE PTSIN, c
270 POKE PTSIN+2, 0
280 VDISYS
290 RETURN
```

Ce programme dessine les épaisseurs disponibles de 1 à 25 pixel. On peut à peine appeler le dernier résultat une 'ligne'. Il s'agit presque d'un rectangle.

APPARENCE DES POINTS TERMINAUX
Set Polyline End Style

Il faudra chercher longtemps avant de trouver un ordinateur dont le BASIC proposera autant de possibilités d'astuces. Avec la routine 108, on peut modifier l'apparence des points terminaux des lignes. Si vous avez essayé le programme précédent, vous aurez remarqué que les points terminaux des lignes étaient 'coupés' (ou 'rectangulaires'). C'est l'apparence normale lorsque l'on dessine une ligne. Mais on peut également avoir des points terminaux ronds. Ou bien encore, ce qui est très pratique pour des applications scientifiques ou techniques, on peut leur donner l'apparence d'une flèche.
Essayez de le faire en BASIC. Le calcul est important et dure assez longtemps pour des lignes épaisses. Mais le BASIC ATARI et le VDI résolvent ce problème.

Nous pouvons définir l'épaisseur de la ligne ainsi que l'apparence de la fin et du début de la ligne (séparément).
Bien que cette fonction ne soit pas applicable tous les jours dans chaque programme, elle peut alléger certains travaux.

Ces remarques concernent également les instructions CIRCLE et ELLIPSE. C'est vrai en particulier pour les arcs de cercle ou d'ellipses. Vous pouvez modifier le programme donné en exemple afin de dessiner un arc de cercle à la place de l'instruction LINEF (lignes 140 à 180).

Les paramètres sont l'apparence du début et de la fin de la ligne et se trouvent dans INTIN et INTIN+2. On peut choisir les valeurs 0, 1 et 2. Le chiffre 0 correspond à la forme rectangulaire, 1 à une flèche et 2 à un rond.

```
100 debut=0 : fin=0
110 GOSUB set.end
120 i=20
130 FULLW 2: CLEARW 2
140 LINEF 20, c+i, 500, c+i
150 i=i+24
160 FOR c=3 TO 15 STEP 2
170 GOSUB set.width
180 LINEF 20, i, 500, i
190 i=i+35
200 NEXT c
210 c=2 : GOSUB set.width
220 a=inp(2)
230 IF a=27 THEN fin=0 : debut=0 :
    GOSUB set.end : END
240 fin=fin+1
250 IF fin=3 THEN fin=0 : debut=debut+1
260 IF debut=3 THEN debut=0
270 GOSUB set.end
280 GOTO 120
290 END
300 set.width :
310 POKE CONTRL, 16
320 POKE CONTRL+2, 1
330 POKE CONTRL+6, 0
340 POKE PTSIN, c
350 POKE PTSIN+2, 0
360 VDISYS
370 RETURN
380 set.end:
390 POKE CONTRL, 108
400 POKE CONTRL+2, 0
410 POKE CONTRL+6, 2
420 POKE INTIN, debut
430 POKE INTIN+2, fin
440 VDISYS
450 RETURN
```

Ce programme est écrit à partir du précédent concernant la largeur des lignes.
Vous pouvez donc étendre le précédent plutôt que tout retaper.
Après une exécution, le programme attend que vous appuyez sur une touche autre que ESC. Puis la figure est dessinée avec un nouveau point terminal. ESC permet d'arrêter le programme et de réinitialiser les paramètres.

Nous en avons fini avec les lignes et leurs motifs. Il existe encore d'autres fonctions intéressantes.

TEST DE LA POSITION DE LA SOURIS
Sample Mouse Button State

Lorsqu'on utilise pour la première fois le BASIC du ST, on a l'impression qu'aucune fonction n'a été implémentée pour le test de la souris. Mais le VDI là encore vient à notre secours. La fonction 124 donne la position courante de la souris et indique si une des touches de la souris est pressée. Hormis le CODEOP dans CONTRL, cette fonction ne nécessite aucun paramètre.

Après l'appel par VDISYS, INTIN contient une valeur indiquant si une touche est pressée. Si la valeur est 0, aucune touche n'est pressée. Si la valeur est 1, la touche gauche est pressée, avec 2, c'est la touche droite. Si les deux touches sont pressées en même temps, INTIN a pour valeur 3. PTSOUT contient la position de la souris en abscisse. Cette valeur est comme toutes les positions données par le VDI, indiquée par rapport aux coordonnées de l'écran et non par rapport à celles d'une fenêtre. PTSOUT+2 contient l'ordonnée de la souris.

Nous avons écrit un programme un peu plus long avec la fonction 124. Il montre comment utiliser la souris pour construire des menus. Nous n'avons pas lésiné sur les trucs, aussi il serait bon que vous analysiez ce programme avec attention.

```
10  a$(1)="chargement du programme"
20  a$(2)="mise en route du programme"
30  a$(3)="sauvegarde des données"
40  a$(4)="fin du programme"
50  p(1)=7:p(2)=8:p(3)=9:p(4)=10
60  actif=1: desactive=2
70  FULLW 2: CLEARW 2
80  GOTOXY 5,5: ?"choisissez :"
90  effet=desactive: GOSUB text.effet: GOSUB 210
100 GOSUB mouse.button
110 choix=int((y.pos-108)/16): REM GOTOXY 1,1: ? y.pos
120 GOSUB mouse.on: If button=0 THEN 100
130 GOSUB mouse.off
140 IF choix<1 OR choix>4 THEN 90
150 GOSUB 210
160 effet=actif: GOSUB text.effet
170 GOTOXY 5,p(choix): ?a$(choix)
180 IF choix <>4 then effet=desactive ELSE effet=0
190 GOSUB TEXT.effet
200 IF choix=4 THEN END ELSE 100
210 FOR i=1 TO 4
220 GOTOXY 5, p(i): ?a$(i)
230 NEXT i
240 RETURN
250 GOTO 100
260 '
270 mouse.on: REM ******************
280 POKE CONTRL, 122
290 POKE CONTRL+2, 0
300 POKE CONTRL+6, 1
310 POKE INTIN, 0
320 VDISYS
330 RETURN
340'
350 mouse.off: REM ****************
360 POKE CONTRL, 123
370 POKE CONTRL+2, 0
```

```
380 POKE CONTRL+6, 0
390 VDISYS
400 RETURN
410'
420 mouse.button: REM ***************
430 POKE COCNTRL, 124
440 POKE CONTRL+2, 0
450 POKE CONTRL+6, 0
460 VDISYS
470 button=PEEK(INTOUT)
480 x.pos=PEEK(PTSOUT)
490 y.pos=PEEK(PTSOUT+2)-38
500 RETURN
510'
520 text.effet: REM ******************
530 POKE CONTRL, 106
540 POKE CONTRL+2, 0
550 POKE CONTRL+6, 1
560 POKE CONTRL+10, 1
570 POKE INTIN, effet
580 VDISYS
590 RETURN
```

Après la mise en route du programme apparaît à l'écran un petit menu. Avec la souris, vous pouvez vous positionner sur un des choix du menu et le choisir en cliquant.
Les trois premiers points ne servent à rien dans l'exemple. Mais si vous choisissez le quatrième point, le programme est arrêté.
Pour mettre en valeur le point du menu qui est choisi,celui-ci est affiché en gras alors que les autres points sont en affichage contrasté. Cette présentation est faite en ligne 60. Choisissez les variables 'actif' et 'déactive' selon votre préférence.
Pour le choix, seule l'ordonnée est nécessaire dans notre cas. La valeur obtenue à partir du VDI est transformée en affichage en ligne 110. Vous devrez y faire les adaptations nécessaires pour vos applications.

Afin de faire correspondre plus facilement l'ordonnée, il est conseillé de supprimer l'instruction REM en ligne 110. Dans ce cas, l'ordonnée sera affichée dans le coin en haut à gauche.

ACTIVATION DU MODE ECRITURE
Set Writing Mode

Cette fonction étend les possibilités de l'affichage. Avec la fonction 32, on peut modifier le mode d'affichage. Jusqu'ici, nous n'avons fait des affichages qu'en mode REPLACE. Si quelque chose était affiché à l'écran on ne pouvait qu'écrire par dessus ou bien l'effacer. Pour certaines applications, ce peut être un inconvénient. Le mode transparent par exemple n'efface pas le fond lors d'un affichage. Le mode XOR fonctionne selon le principe du OU EXCLUSIF, qui est une comparaison logique entre deux valeurs. Lorsque l'une des valeurs est 1, le résultat est 1. Si les deux valeurs sont nulles ou égales à 1, le résultat est 0. Dans ce cas, aucun point n'est affiché à l'écran. La dernière possibilité est le mode reverse transparent.

Si vous voulez tester le programme qui suit, il faudra remplacer la ligne 160 par un REM, un " ' " ou bien l'effacer avant la première exécution. Dans ce cas, les affichages se feront sur un écran vide. A la deuxième exécution, vous pourrez faire l'essai avec la ligne 160. Les résultats montrent le rôle des différents modes beaucoup mieux que ne le ferait un long discours.

```
100 FULLW 2: CLEARW 2
110 a$(1)="texte normal, mode replace"
120 a$(2)="texte en mode transparent"
130 a$(3)="Ce texte est en mode XOR"
140 a$(4)="texte reverse transparent"
150 COLOR 1,1,1,2,2
160 FILL 1,1
170 FOR i=1 TO 4
180 GOSUB set.wrt.mode
190 GOTOXY 10,6+i: ?a$(i)
```

```
200 NEXT
210 a=inp(2) : i=1
220 GOSUB set.wrt.mode
230 END
240 set.wrt.mode:
250 POKE INTIN, i
260 POKE CONTRL, 32
270 POKE CONTRL+2, 0
280 POKE CONTRL+6, 1
290 VDISYS
300 RETURN
```

1.2.3 L'INSTRUCTION GEMSYS

Cette instruction joue un rôle semblable à l'instruction VDISYS. Elle appelle L'AES de GEM. Cependant, les routines de GEMSYS se différencient nettement des routines disponibles avec VDISYS.
Pour appeler l'AES, il faut également préciser des paramètres dans certains vecteurs. Les paramètres des vecteurs de l'AES sont différents de ceux du VDI.
De plus, les adresses des 6 vecteurs ne sont pas toujours utilisables. A la place, on dispose d'une table avec les adresses des vecteurs. Cette table peut être manipulée avec la variable BASIC réservée **GB**.
Cette table est constituée de 24 octets ou de 6 adresses de 32 bits. Les vecteurs de l'AES portent les noms plus ou moins fixés : CONTROL, GLOBAL, INT.IN, INT.OUT, ADDR.IN et ADDR.OUT. Ces noms ont été fixés par DIGITAL RESEARCH. Nous nous tiendrons à ces noms pour la description des vecteurs.

Le vecteur CONTROL fonctionne comme le vecteur CONTROL du VDI. Mais le programmeur BASIC n'a pas besoin de s'occuper de ce vecteur car il est géré par le BASIC lors de l'appel de GEMSYS.
La seconde adresse de la table GB pointe sur le vecteur GLOBAL. Ce vecteur contient certains paramètres qui ne doivent pas être modifiés. Ces valeurs sont fixées par GEM.

Les vecteurs restant fonctionnent comme leurs homologues du VDI. Il faut remarquer que les éléments du vecteur INT doivent être précisés sous forme de mots (16 bits). Les éléments du vecteur ADDR sont en mots longs car il faut préciser des adresses pour plusieurs appels.

Les fonctions de l'AES sont ainsi faites que beaucoup d'applications ne seront pas possibles en BASIC, sauf au prix d'énormes efforts de programmation. Les aspects de l'AES concernant le clavier ou la souris sont gérés par des routines d'interruption très complexes. Il est donc impossible de s'en servir en BASIC.
Si vous essayez, vous serez confrontés à de multiples plantages.

Si la manipulation de l'AES vous intéresse vraiment, il vous faudra apprendre un langage tel que C, PASCAL ou MODULA2 (à moins que ce ne soit déjà fait).
Seuls ces langages permettent de manipuler les fonctions de l'AES assez rapidement.
Il serait bien sûr intéressant de pouvoir utiliser certaines des possibilités en BASIC. On pourrait par exemple changer le nom de la fenêtre OUTPUT. Nous avons trituré notre matière grise (et la touche RESET!) sans cesse jusqu'à trouver un truc. Le voici dans le programme qui suit:

```
10   GOSUB gem.arrays
20   x1=0:a$="fenêtre d'affichage"
30   POKE INT.IN,3
40   POKE INT.IN+2,2
50   x1=VARPTR(a$)
60   POKE INT.IN+4,x1 / 2^16
70   POKE INT.IN+6,x1 and &HFFFF
80   POKE INT.IN+8,0
90   POKE INT.IN+10,0
100  GEMSYS 105
110  END
120  '
50000 gem.arrays :
```

```
50003 INT.IN=PEEK(gb+8)*2^16+PEEK(gb+10)
50007 RETURN
```

Comment ça marche? dans le sous programme 'gem.arrays', on calcule en ligne 50003 l'adresse du vecteur INT.IN et on la sauvegarde dans la variable portant son nom. La variable xl doit être déterminée à ce point afin de ne pas fausser le résultat de la fonction VARPTR plus tard. La variable a$ contient le texte pour la fenêtre d'affichage. Il ne faudra pas écrire un roman car le texte ne tiendrait pas dans la place disponible. La longueur du texte ne doit pas dépasser 20 caractères.
Enfin, les paramètres nécessaires sont mis dans le vecteur INT.IN à l'aide de POKE. C'est en cela que l'instruction GEMSYS se différencie de l'instruction VDISYS. Dans les lignes 60 et 70, nous donnons au vecteur la valeur de l'adresse de la chaîne. Elle est obtenue avec la fonction VARPTR (ligne 50). Si vous ne comprenez pas bien comment ce paramètre est précisé, vous devriez vous reporter dans le chapitre précédent au paragraphe traitant de la fonction VARPTR.

Il est intéressant de noter que nous ne donnons aucune valeur au vecteur CONTROL. Nous indiquons directement le numéro de la fonction avec GEMSYS. Le BASIC calcule les valeurs pour le vecteur CONTROL à partir de ce numéro. Il serait bon de pouvoir disposer de ce 'luxe' avec VDISYS.

1.3. LA RAPIDITE DU BASIC

Naturellement, nous avons désiré connaître la vitesse de traitement du BASIC.
Mais comment calculer la vitesse du BASIC alors qu'on ne dispose d'aucune fonction pour connaître le temps en BASIC. Le problème heureusement peut être résolu de manière relativement simple. Le système d'exploitation possède des compteurs gérés par interruptions aux adresse $4BA et $4BD. Le contenu de ces cases mémoires est donné sous forme D'un mot long, c'est à dire sur 32 bits. Le mot long est incrémenté de un 200 fois par seconde.

Cela correspond à une durée de 5 millisecondes. Mais la plupart des instructions sont exécutés en moins de temps. Pour cette raison, on exécutera une instruction plusieurs fois dans une boucle et on divisera le résultat obtenu par le nombre d'exécutions. Après avoir soustrait le temps nécessaire à la boucle FOR NEXT, on obtient le temps nécessaire pour l'exécution d'une instruction.
Nous avons obtenu le temps de diverses instructions avec le petit programme suivant :

```
100 timer=&H4BC
110 temps1=PEEK(timer)
120 FOR i=1 to 10000
130 LET a=1
140 NEXT i
150 temps2=PEEK(timer)
160 temps=temps2-temps1
170 temps=(temps*5/10000)-.8495
180 ? "l'instruction en ligne 130 nécessite "temps"
    millisecondes"
```

En moyenne, les instructions du BASIC durent de 0.6 à 1.9 millisecondes. La lanterne rouge est l'instruction PRINT.
L'affichage d'un seul caractère dure environ 4,5 millisecondes. Cependant le calcul précis de la durée d'une instruction PRINT n'est pas facile. Nous avons dû calculer la durée de l'instruction GOTOXY. Puis nous avons noté la durée de la ligne:

```
130 GOTOXY 0,0:?"a";
```

Et nous avons soustrait du résultat la durée de l'instruction GOTOXY. Faites attention au point virgule après l'instruction PRINT. Sans le point virgule, les caractères CR et LF sont exécutés eux aussi. Le temps qu'ils nécessitent n'a rien à voir avec la durée de l'instruction PRINT.
Les durées deviennent catastrophiques si on supprime l'instruction GOTOXY et le point virgule. L'écran subit alors un scrolling à (presque) chaque affichage.

Si vous voulez tester la durée pour cela, réduisez le nombre d'exécutions de la boucle à 100 ou 200 (lignes 120 et 170). Sinon, le test serait trop long.
Les résultats les plus intéressants sont donnés par les fonctions à virgule flottantes comme SQR, SIN ou LOG. Ces fonctions sont exceptionnellement rapides. Si on compare le temps de traitement avec celui des autres ordinateurs, la différence est incroyable. La fonction SQR du C64 dure environ 54 millisecondes, celle du CPC dure environ 27 millisecondes. Le ST, lui, met environ 1 milliseconde !
Ces durées exceptionnelles tiennent au fait que le BASIC a été écrit en grande partie en C. Seules les fonctions à virgule flottante ont été écrites en assembleur. C'est la firme MOTOROLA qui est l'auteur des fonctions à virgule flottante. En tant que développeur du 68000, elle savait comment tirer parti entièrement de sa rapidité.

1.4. BASIC ET LANGAGE MACHINE

Vous vous demandez peut être s'il est vraiment utile de mélanger BASIC et langage machine. On peut presque tout faire en BASIC. C'est justement ce 'presque' qui nous a poussé à écrire de petites routines en langage machine à appeler du BASIC. Le premier problème était l'horloge. Nous avons crée une horloge très précise avec le système d'exploitation. Cette horloge a un pas de 2 secondes. Mais on ne peut pas obtenir l'heure avec le BASIC. Il fallait un remède à cela.

1.4.1. Les endroits possibles pour le langage machine.

Lorsqu'il s'agit d'afficher l'heure avec des programmes BASIC, on s'occupe d'abord du moyen de contrôle. Puis on peut lire l'heure indiquée (INPUT). Il faut ensuite la saisir de nouveau. Cela n'est pas possible pour des raison de durée. Le problème est vite résolu en langage machine. Mais se pose alors la question de savoir comment exécuter ce programme en même temps que le BASIC sans problème.

Nous avons trouvé plusieurs solutions que nous allons vous expliquer.
La solution la plus simple est sans doute d'utiliser la place mémoire libre après l'écran. L'organisation de la mémoire du ST est ainsi conçue que la RAM d'écran se situe dans les 32 Koctets supérieurs de la mémoire. Pour le 520ST et le 260ST, il s'agit de l'adresse $78000. Pour le 520ST+ (1 méga), c'est l'adresse $F8000. L'écran n'utilise que 32000 octets (=640*400 bits). Les 768 octets restant ne sont jamais utilisés par le système d'exploitation. On peut y inscrire des programmes si leur longueur convient.
Cependant, certains problèmes se posent avec cette solution. Chaque programme qui utilise cet emplacement doit savoir s'il s'agit d'un appareil avec 512 ou 1024K et choisir l'adresse en conséquence. Il faudra résoudre ce problème. Mais il existe une autre raison pour ne pas utiliser ce secteur. Lorsqu'un ordinateur dispose d'une telle zone mémoire, il est à parier que de nombreux programmes seront écrits de façon à y être installés. Plus tard, si deux programmes doivent occuper ce même emplacement, les problèmes se feront sentir.

Où installer la routine alors ? Il existe un vieux truc utilisable aussi sur le ST. On la met tout simplement dans une chaîne de caractères.
L'interpréteur BASIC se moque de savoir que la chaîne A$ contienne le texte 'Hegel: to be=to do; Sartre: to do=to be; Sinatra=to be do be do' ou bien que la chaîne 'BASTON' contienne un programme en langage machine. Démontrons cela de façon concrète. En exemple, nous créerons une fonction qui donne l'heure. Il nous faudra d'abord écrire le programme en assembleur qui convient.

En langage machine, on peut obtenir l'heure d'une manière très simple. Si la valeur $2C est mise sur la pile et qu'on exécute l'instruction TRAP #1, le registre D0 contient l'heure.
Malheureusement, l'heure n'est pas exprimée en clair dans D0. Elle est divisée en bits et doit être décodée. Nous montrerons plus tard comment le faire.

```
000000      MOVE.L  A0,A5           adresse de la routine
                                    après A5
000002      MOVE.W  #$2C,-(A7)      charge numéro de
                                    fonction de l'heure
000004      TRAP    #1              exécute fonction
000008      ADDQ.L  #2,A7           réinitialise stackpointer
00000A      MOVE.W  D0,$10(A5)      écrire heure dans
                                    mémoire
00000E      RTS
000010      DS.W    1               place pour temps
```

Lorsque ce programme est assemblé dans la mémoire et est appelé du BASIC avec une instruction CALL, l'adresse de la routine se trouve dans le registre A0.
Nous sauvegardons cette adresse dans le registre A5 car nous en aurons besoin plus tard. Ensuite, nous mettons le numéro de fonction sur la pile et nous exécutons l'instruction TRAP. Après le TRAP, nous devons réinitialiser le registre A', qui est le pointeur de pile, car le numéro de fonction se trouve toujours sur la pile.
Maintenant, l'heure obtenue dans D0 se trouve dans la mémoire. Dans le registre A5, nous avons sauvegardé l'adresse de la première instruction. Or la mémoire utilisée pour le temps est l'adresse qui se trouve à 16 octets du début. (qui est dans A5). Cette adresse se trouve exactement après l'instruction RTS qui termine le programme.
Quand le programme est assemblé, on obtient des CODEOPs, c'est à dire des nombres hexadécimaux qui sont exécutés comme programme. dans notre exemple, les codeop sont :

$2A, $48, $3F, $3C, $00, $2C, $4E, $41,
$54, $8F, $3B, $40, $00, $10, $4E, $75

Ces valeurs doivent être mises dans une chaîne de caractères. On peut par exemple utiliser le programme suivant :

```
10  FOR i=0 TO 17
20  READ byte
30  heure$=heure$+CHR$(byte)
```

```
40  NEXT i
50  DATA&H2A,&H48,&H3F,&H3C,&H00,&H2C,&H4E,&H41
60  DATA&H54,&H8F,&H3B,&H40,&H00,&H10,&H4E,&H75
70  DATA &HFF,&HFF
```

Les deux dernières valeurs sont particulièrement importantes. C'est là que sera inscrite l'heure par la suite. La place nécessaire pour le résultat doit évidemment être réservée dans la chaîne. Autrement il ne serait pas possible de le lire. La première partie est donc terminée. Notre chaîne est prête.
Maintenant, il nous faut trouver un moyen de mettre en route la routine. Il faut d'abord savoir où se trouve la chaîne en mémoire. nous utiliserons son adresse comme paramètre de l'instruction CALL. On l'obtiendra avec la fonction VARPTR qui donne l'adresse du descripteur de chaîne (voir description de VARPTR dans le chapitre sur le BASIC.). Dans les octets 3 à 6 du descripteur de chaîne se trouve l'adresse de la chaîne c'est à dire du programme d'horloge.

```
80  adresse=0
90  adresse=PEEK(VARPTR(heure$)+2)
```

Notons une particularité : la fonction PEEK donne sur 32 bits l'adresse de la chaîne se trouvant dans la variable 'adresse'.
Normalement le résultat devrait être sur 16 bits. Mais si l'adresse lue est le résultat de la fonction VARPTR, le BASIC charge deux valeurs sur 16 bits, ce qui est très utile.
La ligne 70 est indispensable. Si on n'initialisait pas la variable adresse, le résultat pourrait être modifié lors de l'initialisation avec la fonction VARPTR.
Maintenant, nous pouvons en venir au fait et lire l'heure. Pour cela, il faut utiliser l'instruction CALL.

```
100 CALL adresse
```

Après le call, l'heure se trouve dans les deux derniers caractères de la chaîne "heure$". On peut isoler cette partie avec RIGHT$ et calculer l'heure.

```
110 TIME$=RIGHT$(heure$,2)
```

Ce système vous paraît peut être surprenant. Il fonctionne, mais il a aussi des inconvénients. Le plus gênant est que la taille de la routine est limitée à 256 octets. C'est la longueur maximale de la chaîne. De plus, il faut parfois fixer des paramètres à la place des caractères. Mais il est très difficile de fixer des paramètres pour une routine. La troisième méthode que nous voulons vous présenter évite tous ces problèmes.

Elle est la plus flexible pour relier des programmes en langage machine avec du BASIC. La routine est mise dans un vecteur INTEGER. Si on observe la structure d'un vecteur INTEGER dans la mémoire du ST, il apparaît que les éléments du vecteur sont placés les uns à la suite des autres.
L'élément dont l'indice est le plus faible se trouve à l'adresse la plus petite. Chaque élément tient sur 2 octets, ce qui correspond aux CODEOPs du 68000. La taille d'un programme en langage machine inscrit dans un vecteur n'est pas aussi limitée qu'avec la méthode des chaînes. Les programmes peuvent faire 1000 octets et même plus.

```
10  DIM horloge%(8)
20  FOR i=0 TO 8
30  READ horloge%(i)
40  NEXT i
50  DATA &H2A48,&H3F3C,&H002C,&H4E41,&H548F
60  DATA &H3B40,&H0010,&H4E75,&H0000
```

Comme vous le voyez, nous avons dimensionné le vecteur (ligne 10) et nous y avons installé le programme. Cette initialisation elle même est plus courte qu'avec le programme précédent car les éléments de DATA tiennent sur 16 bits.
Ici aussi, il nous faut fixer l'adresse de la routine d'horloge avec la fonction VARPTR. C'est très simple. Le résultat de la fonction VARPTR pointe directement sur la première instruction de la routine. Nous pouvons donc l'utiliser directement comme adresse de branchement pour l'instruction CALL!

```
70  ad=0
80  ad=VARPTR(horloge%(0))
90  CALL ad
```

Nous pouvons obtenir le résultat très facilement. Il se trouve dans l'élément horloge%(8).

```
100 ? horloge%(8)
```

On peut donner les paramètres de la routine de la même manière. Il suffit d'inscrire les paramètres dans les vecteurs correspondants et le programme les charge à partir des cases mémoires choisies. Si ce n'est pas de la flexibilité alors qu'est-ce?


```
70  ad=0
80  ad=VARPTR(horloge%(0))
90  CALL ad
```

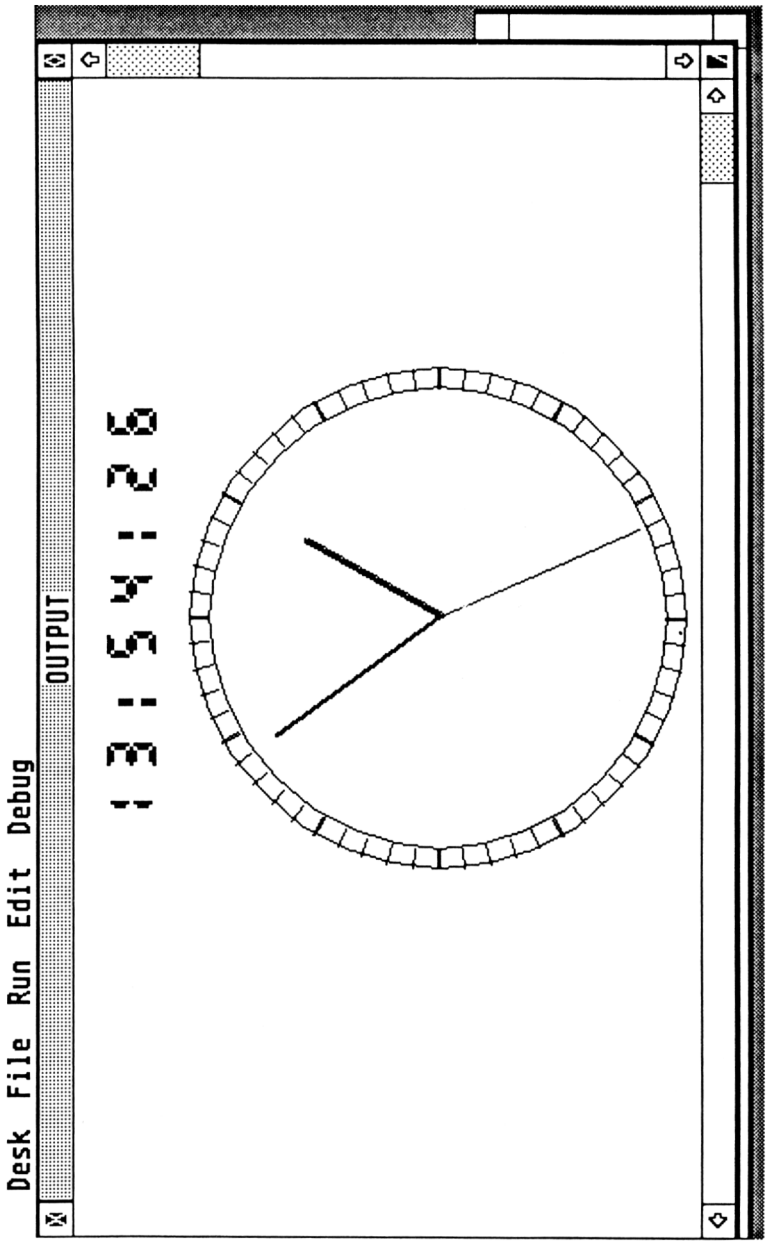
Nous pouvons obtenir le résultat très facilement. Il se trouve dans l'élément horloge%(8).

```
100 ? horloge%(8)
```

On peut donner les paramètres de la routine de la même manière. Il suffit d'inscrire les paramètres dans les vecteurs correspondants et le programme les charge à partir des cases mémoires choisies. Si ce n'est pas de la flexibilité alors qu'est-ce?

1.5. L'HORLOGE LA PLUS CHERE DE VOTRE FOYER

Il y a des gens qui penseront que ce qui suit est superflu. Nous ne sommes pas du même avis. Naturellement, votre ST ne doit pas seulement servir d'horloge. Il serait sous-utilisé ! Mais avec un tel programme, on peut montrer certaines techniques de programmation de base.
Nous nous sommes servi des nombreuses possibilités graphiques du ST pour le programme qui suit. Et nous avons écrit un programme amélioré pour lire l'heure. Mais avant de l'étudier, le voici :


Desk File Run Edit Debug
OUTPUT
13:54:26

```
1000  h0=25
1010  dim hor%(23)
1020  xm% = 320:ym% = 200
1030  sec.z% = 115 : min.z% =105 : hrs.z% = 80
1040  pi=4*atn(1)
1050  g89 = 89.9*(pi/180) : g90 = pi/2 : g91 = 90.1*(pi/180)
1060  hor  = 0
1070  '
1080  fullw 2:clearw 2
1090  gosub hauteur
1100  gosub init.hor
1110  gosub cadran
1120  '
1130  loop: '   *************************
1140  hor = varptr (hor%(0))
1150  call hor
1160  if sec% = hor%(20)*2 then loop
1170  '
1180  ' efface l'aiguille****************
1190  sec% = hor%(20)*2
1200  color 0,0,0,0,0
1210  dmy% = hrs% : hrs% = hor%(22)
1220  if dmy%<> hrs% then phi = phihrs :r%=hrs.z%:gosub aiguille
1230  dmy% = min% : min% = hor%(21)
1240  if dmy%<> min% then phi = phimin :r%=min.z%:gosub aiguille
1250  phi = phisec : r%=sec.z%: gosub aiguille
1260  '
1270  '
1280  ' redessine l'aiguille ************
1290  color 1,0,1,1,1
1300  phisec = sec%*pi/30-g90: r%=sec.z%
1310  phi = phisec : gosub aiguille
1320  phimin = min%* 6 * (pi/180)-g90 : r%=min.z%
1330  phi = phimin : gosub aiguille
```

```
1340  phihrs = hrs%* 30 * (pi/180)-g90 : r%=hrs.z%
1350  phi = phihrs : gosub aiguille
1360  gosub digital
1370  goto loop
1380  '
1390  aiguille: ' ************************
1400  linef xm%  ,ym%  ,xm%+r%*cos(phi)  ,ym%+r%*sin(phi)
1410  if r%= sec.z% then return
1420  '
1430  linef xm%+1,ym%  ,xm%+r%*cos(phi)+1,ym%+r%*sin(phi)
1440  linef xm%  ,ym%+1,xm%+r%*cos(phi)  ,ym%+r%*sin(phi)+1
1450  linef xm%+1,ym%+1,xm%+r%*cos(phi)+1,ym%+r%*sin(phi)+1
1460  if r%= min.z%then return
1470  '
1480  linef xm%+2,ym%  ,xm%+r%*cos(phi)+2,ym%+r%*sin(phi)
1490  linef xm%  ,ym%+2,xm%+r%*cos(phi)  ,ym%+r%*sin(phi)+2
1500  linef xm%+2,ym%+2,xm%+r%*cos(phi)+2,ym%+r%*sin(phi)+2
1510  return
1520  '
1530  cadran: '*************************
1540  circle xm%,ym%,120,120
1550  circle xm%,ym%,130,130
1560  for hrs%= 1 to 12
1570  phi = hrs%* 30 * (pi/180)-g90 : r1%=130 : r0%=120
1580  linef xm%+r0%*cos(phi),ym%+r0%*sin(phi),xm%+r1%*cos(phi),ym%+r1%*sin(phi)
1590  phi = hrs%* 30 * (pi/180)-g89 : r1%=130 : r0%=120
1600  linef xm%+r0%*cos(phi),ym%+r0%*sin(phi),xm%+r1%*cos(phi),ym%+r1%*sin(phi)
1610  phi = hrs%* 30 * (pi/180)-g91 : r1%=130 : r0%=120
1620  linef xm%+r0%*cos(phi),ym%+r0%*sin(phi),xm%+r1%*cos(phi),ym%+r1%*sin(phi)
1630  next hrs%
1640  for min%= 1 to 59
1650  phi = min%* (pi/30) : r1%=130 : r0%=120
1660  linef xm%+r0%*cos(phi),ym%+r0%*sin(phi),xm%+r1%*cos(phi),ym%+r1%*sin(phi)
1670  next min%
```

```
1680 return
1690 '
1700 digital: ' ************************
1710 sec$=str$(sec%): if len(sec$)=2 then sec$=" 0"+right$(sec$,1)
1720 min$=str$(min%): if len(min$)=2 then min$=" 0"+right$(min$,1)
1730 hrs$=str$(std%): if len(hrs$)=2 then hrs$=" 0"+right$(hrs$,1)
1740 timdig$=right$(hrs$,2)+"Z"+right$(min$,2)+"Z"+right$(sec$,2)
1750 gosub printdig
1760 return
1770 '
1780 hauteur: ' ************************
1790 poke contrl    ,107
1800 poke contrl+2,0
1810 poke contrl+6 ,1
1820 poke intin,h0
1830 vdisys
1840 return
1850 '
1860 printdig: rem *********************
1870 poke contrl    ,11
1880 poke contrl+2 ,2
1890 poke contrl+6 ,10
1900 poke contrl+10,10
1910 poke contrl+12,2
1920 poke intin     ,1
1930 poke intin+2   ,1
1940 for i%=1 to 8
1950 poke intin + (i%*2+2),asc(mid$(timdig$,i%,1))-32
1960 next i%
1970 poke ptsin,210
1980 poke ptsin+2,80
1990 poke ptsin+4,220
2000 poke ptsin+6,0
2010 vdisys
```

```
2020  return
2030  '
2040  init.hor:   ' **********************
2050  data &h2a48,&h3f3c,&h002c,&h4e41,&h548f,&h3b40,&h0028,&h026d
2060  data &h001f,&h0028,&hea48,&h3200,&h0240,&h003f,&h3b40,&h002a
2070  data &hec49,&h3b41,&h002c,&h4e75,&h0000,&h0000,&h0000,&h0000
2080  for i% = 0 to 23
2090  read hor%(i%)
2100  next i%
2110  return
```

Dans les sept premières lignes, plusieurs variables sont initialisées.

h0 est la hauteur des caractères pour l'affichage digital. Si vous arrêtez le programme, la taille normale n'est pas automatiquement remise en place. C'est ennuyeux si le programme ne fonctionne pas et que vous devez corriger des fautes de frappe. Mettez cette variable à 10 lors du test du programme, vous travaillerez alors sous l'affichage normal.

xm%, ym% indiquent le milieu de la page.

sec.z%, min.z%, std.z% indiquent la longueur des trois pointeurs.

pi est le nombre pi (3.1415927) qui n'est pas disponible directement en BASIC ST.

Ensuite on fixe la hauteur, on précise le vecteur du programme en langage machine et on dessine l'horloge.
Le programme principal commence en ligne 1130. Le programme en langage machine met les secondes dans **heure%(20)**, les minutes dans **heure%(21)** et les heures dans heure%(22). Après un branchement, la nouvelle valeur des secondes est comparée avec l'ancienne. Tant qu'elles sont égales, la boucle est répétée.
Lorsque les secondes sont modifiées, la boucle est quittée.
Le reste du programme est sans problème. Avec quelques connaissances en BASIC, vous devriez le comprendre sans difficultés.

1.6. HARDCOPY AUTOMATIQUE

Pour résoudre ce problème, il existe au moins deux solutions toutes prêtes. la première consiste à appeler une routine VDISYS précise. Il faut d'abord préciser les paramètres suivants dans les vecteurs avec POKE:

```
CONTRL, 5
CONTRL+ 2, 0
CONTRL+ 6, 0
CONTRL+10, 17
```

Mais il existe un moyen plus simple. Un seul POKE est nécessaire:

```
POKE 1262, 0
```

La case mémoire en question contient normalement la valeur -1. Si vous pressez simultanément les touches ALTERNATE et HELP, le contenu est incrémenté de 1, ce qui exécute le HARDCOPY. Cela peut bien sûr être simulé simplement avec le POKE indiqué.
Le HARDCOPY s'arrête si vous pressez de nouveau les touches ALT et HELP.

CHAPITRE 2

UTILITAIRES POUR L'ATARI 520ST

Dans ce chapitre, nous vous donnons de nombreux petits programmes qui pourront vous aider. Il s'agit pour la plupart de programmes en langage machine qui étendent ou bien modifient les fonctions du système d'exploitation. Ils sont donc inscrits à coté du système d'exploitation en mémoire et vous faciliteront (ou vous rendront possible) le travail avec l'ordinateur ou bien avec d'autres logiciels.

Les différentes parties contiennent une courte explication du programme, de ses possibilités et de son utilisation. Vous trouverez ensuite un programme source en assembleur 68000. Les commentaires devraient vous permettre d'adapter ou d'étendre le programme en fonction de vos besoins. (à condition que vous possédiez un assembleur). Afin que ceux qui ne peuvent programmer qu'en BASIC ne soient pas laissés pour compte, vous trouverez également des chargeurs BASIC. Ces programmes contiennent le programme en langage machine sous forme de DATAs. Les tests de somme vous permettront de détecter des erreurs de saisie avec une bonne fiabilité. Si vous démarrez ces programmes à partir du BASIC, ils créent automatiquement un fichier programme tel que vous pourriez L'obtenir avec un assemblage et linkage. Après cette exécution, vous pourrez mettre en route ce programme en cliquant sur l'icône correspondant avec la souris.

2.1. L'HEURE A TOUT MOMENT

Dans le programme que nous voulons vous présenter, vous rencontrerez trois trucs que vous pourrez reprendre pour vos propres programmes. D'abord, il s'agit d'exécuter un programme périodiquement grâce aux interruptions. Ensuite, vous verrez comment écrire un petit programme en mémoire sans qu'il puisse être écrasé par d'autres.

Enfin, nous vous montrerons comment manipuler directement le système de l'ATARI.

Le programme crée une horloge digitale qui demeurera dans le coin droit en haut de l'écran. Ce programme est particulièrement utile si vous l'utilisez avec des logiciels sous GEM ou avec le desktop. Dans ces cas, la première ligne sert comme ligne de statut et les dix derniers caractères sont libres, ce qui n'est pas toujours le cas avec le TOS.

Comme nous l'avons indiqué au début, l'heure doit être initialisée. Pour cela, nous utilisons **le Vertical Blank Interrupt (VBL)** de l'ordinateur, qui est appelé chaque fois que l'ordinateur a affiché une page écran complète. Cela se produit 70 fois par seconde pour un moniteur monochrome. Dans la routine VBL, l'ordinateur teste une liste de branchement constituée normalement de 8 données, qui contiennent les adresses des routines-utilisateurs devant être exécutées dans le VBL. Une adresse égale à zéro signifie que que la donnée correspondante est vide. Pour exécuter notre routine, il faut parcourir la liste jusqu'à ce qu'on trouve un nul. On peut alors y inscrire l'adresse de notre propre routine VBL qui sera donc exécutée à chaque interruption VBL.

Mais où inscrire son programme de manière à ce qu'il ne soit pas écrasé par d'autres logiciels mis en route après l'initialisation?

Si le programme ne fait pas plus de 3 pages ($300 = 768 octets), on peut l'écrire après la mémoire écran. L'ATARI ST réserve à la mise en route les 32 Koctets supérieurs de la mémoire pour l'écran. Pour une machine de 512K, l'écran se trouve de $78000 à $7FFFF. Si vous faites le calcul, vous trouverez que 640*400 font 256000 bits, soit 32000 octets. Or 32 Koctets font 32768 octets, si bien qu'il reste 768 octets de libre. La mémoire d'écran ne se sert donc que des adresses $78000 à $7FCFF. Le secteur $7FD00-$7FFFF n'est pas modifié par l'affichage à l'écran ni quand on l'efface. Ce secteur est l'endroit idéal pour de petites routines qui doivent rester en mémoire.

Un programme d'initialisation devra donc copier le programme derrière l'écran et fixer le vecteur VBL à cette adresse. Cela est effectué au début de notre programme.
Etant donné que nous devons avoir accès aux variables système, il faut d'abord activer le mode superviseur. Puis la liste VBL est testée jusqu'à ce qu'une place libre soit trouvée. L'adresse de cette donnée est mise dans A2. La longueur de la routine, que nous voulons copier, sert de compteur. Nous prendrons pour adresse cible le début de l'écran auquel nous ajouterons 32000, c'est à dire la longueur de l'écran. Puis nous copierons le programme dans le secteur mémoire ainsi calculé. Ensuite, nous appelons la routine pour l'initialisation de l'affichage du temps. Enfin, nous faisons pointer le vecteur VBL encore une fois sur notre routine, et nous retournons au desktop.

Pour la routine chiffres on appelle la routine line-A qui positionne un pointeur dans A1 sur un vecteur des trois motifs-systèmes. Puis nous récupérons l'adresse du motif numéro 2, c'est à dire le motif 8*16 qui est l'écriture standard pour les moniteurs monochromes. ensuite nous mettons notre compteur VBL à 1 afin que notre routine d'affichage soit mise en route après le VBL suivant.

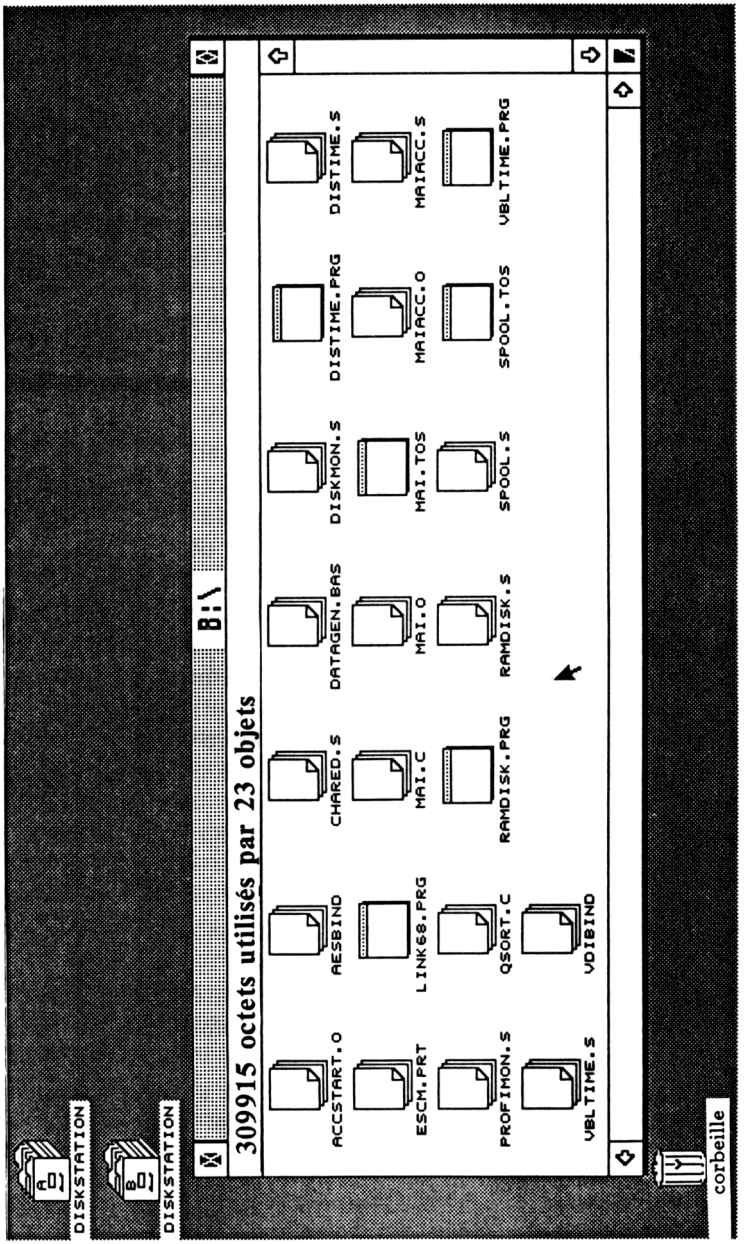
Le compteur est initialisé car il n'est pas possible d'afficher le temps 70 fois par seconde. Après chaque appel, nous décrémentons le compteur et l'heure est affichée seulement quand il est nul. le compteur est de nouveau initialisé par la routine d'affichage. Puis on appelle une routine qui prend le temps sur le processeur de clavier. Elle est identique à la fonction XBIOS correspondante. Le résultat de cette fonction est l'heure sous format DOS avec un pas de 2 secondes. Le processeur de clavier donne le temps en secondes dans le format BCD. Cette information est mémorisée par l'ATARI ST à l'adresse $A46, qui est le label time dans le programme. Nous utilisons cette information pour l'affichage. Les trois octets qui contiennent l'heure, les minutes et les secondes dans le format européen sont affichés par la routine **wrtbcd**, séparés par un signe 'deux points'.

La routine wrtchar écrit un caractère pris dans D0 à la ligne supérieure de l'écran. La position du curseur est prise dans le registre D6 (valeur entre 0 et 79). La position du curseur et l'adresse de base de l'écran permettent de calculer la position courante en RAM graphique. Puis on prend dans l'en-tête l'adresse des données du motif d'écriture, le début de la ligne de RASTER suivante (ligne de bits) ainsi que le nombre de lignes à scruter et la hauteur des caractères. La boucle qui a le label loop, copie ligne de RASTER par ligne de RASTER le caractère directement sur l'écran à partir des données de motif, jusqu'à ce qu'il soit complet. Le programme fonctionne sans adaptation sur un écran monochrome.

Vous vous demanderez peut être pourquoi les caractères ne sont pas tout simplement inscrits à l'écran avec une fonction du BIOS ou de GEMDOS, plutôt que d'être chargés à partir du motif système. La raison est que nous devons manipuler une routine d'interruption. D'une part, un affichage normal risquerait d'être interrompu, le curseur étant quelque part sur l'écran. Nous devrions donc mémoriser la position du curseur, mettre le curseur sur la ligne supérieure, y afficher les caractères, puis remettre le curseur à sa place initiale, ce qui serait long. D'autre part, on peut rencontrer d'autres problèmes si un caractère est justement en train d'être affiché pendant l'interruption ou bien si on utilise un autre motif (sous GEM). Ces problèmes sont évités par la manipulation directe du motif. De plus, c'est un système plus rapide.

L'affichage se fait sur la ligne de statut du desktop.

Sur la page suivante, vous trouverez le listing assembleur du programme. Si vous ne possédez pas d'assembleur, vous pourrez utiliser le chargeur BASIC.
Après avoir été mis en route, le programme crée automatiquement un fichier programme dont le nom est 'time.prg' que vous pourrez démarrer en le cliquant avec la souris.


Bureau Fichier Visualisation Options
14:25:57
DISKSTATION
DISKSTATION
B:\
309915 octets utilisés par 23 objets
ACCSTART.O
AESBIND
CHARED.S
DATAGEN.BAS
DISKMON.S
DISTIME.PRG
DISTIME.S
ESCM.PRT
LINK68.PRG
MAI.C
MAI.O
MAI.TOS
MAIACC.O
MAIACC.S
PROFIMON.S
QSORT.C
RAMDISK.PRG
RAMDISK.S
SPOOL.S
SPOOL.TOS
VBLTIME.PRG
VBLTIME.S
VDIBIND
corbeille

```
*
*       Programme horloge relié à la routine d'interruption
*       le 2/8/85
*

_v_bas_ad       equ $44e    adresse de l'écran
hz_200          equ $4ba    compteur 200 Hz

gemdos          equ 1
setexec         equ 5
bios            equ 13
keep            equ $31
gettime         equ 23
super           equ 38      executer la routine en mode
                            superviseur
xbios           equ 14

                move.l      4(sp),a0        calcul de la taille
                move.l      #$100,d6        du programme
                add.l       12(a0),d6
                add.l       20(a0),d6
                add.l       28(a0),d6
                bsr         init            Initialisation du prog.
                clr         -(sp)
                move.l      d6,-(sp)        Nombre d'octets
                move        #keep,-(sp)
                trap        #gemdos
*
*   Ajouter chaque seconde à l'heure courrante
*
*
init            dc.w        linea
                moveq       #2*4,d0         Numero de caractere
                lea         fontptr(pc),a3
                move.l      (a1,d0),(a3)    Mettre pointeur sur
                                            caractere
                move        #gettime,-(sp)
                trap        xbios
                addq.l      #2,sp           Chercher la duree
                                            actuelle
                move        d0,d7
```

```
                pea     sup_rout(pc)
                move    #super,-(sp)    Executer le reste
                trap    #xbios          en mode superviseur
                addq.l  #6,sp
                rts

sup_rout:
                move    d7,d0
                and     #%11111,d0
                lsl     #1,d0           Secondes en deux
                move    d0,seconde      fois
                move    d7,d0
                lsr     #5,d0
                and     #%111111,d0
                move    d0,minute
                move    d7,d0
                moveq   #11,d1
                lsr     d1,d0
                move    d0,heure
                move    #$2700,sr       interdire interruptions
                move.l  hz_200,time
                add.l   #200,time
                pea     hz_int(pc)
                move    #$45,-(sp)      Vecteur d'interruption
                move    #setexec,-(sp)
                trap    #bios
                addq.l  #8,sp
                move.l  d0,hz_save      Positionner le vecteur
                rts                     200 Hz

hz_int          movem.l d0-d7/a0-a6,-(sp)  Sauve registre
                move.l  time,d0         Deja une seconde ?
                cmp.l   hz_200,d0       Non
                bne     no_show
                add.l   #200,time
                addq    #1,seconde
                cmp     #60,seconde     Incrementer secondes
                bne     show_time
                clr     seconde
                addq    #1,minute
                cmp     #60,minute      Incrementer minutes
```

```
                bne         show_time
                clr         minute
                addq        #1,heure
                cmp         #24,heure     Incrementer heures
                bne         show_time
                clr         heure

show_time:
                moveq       #70,d6        Position curseur
                move        heure,d0      Calcul des heures
                bsr         wrtdez
                bsr         wrtcol
                move        minute,d0     Calcule des minutes
                bsr         wrtdez
                bsr         wrtcol
                move        seconde,d0    Calcul des secondes
                bsr         wrtdez

no_show         movem.l     (sp)+,d0-d7/a0-a6
                move.l      hz_save,-(sp) Adresse de la routine
                rts                       de comparaison

wrtdez:
                move        #$2f,d1       Incrementer dixaine
wrtdez1         addq        #1,d1
                sub         #10,d0
                bpl         wrtdez1
                add         #$3a,d0
                move        d0,-(sp)
                move        d1,d0
                bsr         wrtchar
                move        (sp)+,d0
                bra         wrtchar

wrtcol          moveq       #$3a,d0
*
*    Affichage sur ecran
*
*
adelow          equ         36    Plus petit code ASCII
adehigh         equ         38    Plus grand code ASCII
```

```
cellwd          equ     52      Largeur d'un caractere
fontdat         equ     76      Pointeur sur caractere
formwd          equ     80      Intervalle pour prochaine ligne
formhg          equ     82      Nombre de lignes par caractere
linea           equ     $a000
zeile           equ     80      Octet pro-caractere
*
*       ecrire caractere en RAM graphique
*
*
wrtchar:
                moveq   #0,d1
                move    d6,d1
                addq    #1,d6           curseur sur prochaine
                                        colonne
                move.l  fontptr(pc),a3  chercher indicateur
                add.l   _v_bas_ad,d1    sur caractere
                move.l  d1,a4
                move.l  fontdat(a3),a0  indicateur des donnees
                move    formwd(a3),d2   offset pour prochaine
                move    formhg(a3),d7   ligne de trame
                subq    #1,d7

loop            move.b  (a0,d0),(a4)    ligne de trame sur
                                        ecran
                add     #zeile,a4
                add     d2,a0           indicateur de la
                dbra    d7,loop         prochaine ligne
                rts

fontptr         ds.l    1
hz_save         ds.l    1
seconde         ds.w    1
minute          ds.W    1
heure           ds.W    1
time            ds.l    1
```

```
100  open "R",1,"a:distime.prg",16
110  field#1,16 as bin$
120  a$="": for i=1 to 16: read x$: if x$="*"then 150
130  a=val("&H"+x$): s=s+a:a$=a$+chr$(a): next
140  lset bin$=a$: rec=rec+1: put 1,rec: goto 120
150  data 60,1A,00,00,01,82,00,00,00,00,00,00,00,00,00,00
160  data 00,00,00,00,00,00,00,00,00,00,00,00,20,6f,00,04
170  data 2c,3c,00,00,01,00,dc,a8,00,0c,dc,a8,00,14,dc,a8
180  data 00,1c,61,0a,42,67,2f,06,3f,3c,00,31,4e,41,a0,00
190  data 70,08,47,fa,01,48,26,b1,00,00,3f,3c,00,17,4e,4e
200  data 54,8f,3e,00,48,7a,00,0c,3f,3c,00,26,4e,4e,5c,8f
210  data 4e,75,30,07,c0,7c,00,1f,E3,48,33,c0,00,00,01,78
220  data 30,07,ea,48,c0,7c,00,3f,33,c0,00,00,01,7a,30,07
230  data 72,0b,e2,68,33,c0,00,00,01,7c,46,fc,27,00,23,f9
240  data 00,00,04,ba,00,00,01,7e,06,b9,00,00,00,c8,00,00
250  data 01,7e,48,7a,00,16,3f,3c,00,45,3f,3c,00,05,4e,4d
260  data 50,8f,23,c0,00,00,01,74,4e,75,48,e7,ff,fe,20,39
270  data 00,00,01,7e,b0,b9,00,00,04,ba,66,6a,06,b9,00,00
280  data 00,c8,00,00,01,7e,52,79,00,00,01,78,0c,79,00,3c
290  data 00,00,01,78,66,32,42,79,00,00,01,78,52,79,00,00
300  data 01,7a,0c,79,00,3c,00,00,01,7a,66,1c,42,79,00,00
310  data 01,7a,52,79,00,00,01,7c,0c,79,00,18,00,00,01,7c
320  data 66,06,42,79,00,00,01,7c,7c,46,30,39,00,00,01,7c
330  data 61,20,61,36,30,39,00,00,01,7a,61,16,61,2c,30,39
340  data 00,00,01,78,61,0c,4c,df,7f,ff,2f,39,00,00,01,74
350  data 4e,75,72,2f,52,41,90,7c,00,0a,6a,f8,d0,7c,00,3a
360  data 3f,00,30,01,61,06,30,1f,60,02,70,3a,72,00,32,06
370  data 52,46,26,7a,00,28,d2,b9,00,00,04,4e,28,41,20,6b
380  data 00,4c,34,2b,00,50,3e,2b,00,52,53,47,18,b0,00,00
390  data d8,fc,00,50,d0,c2,51,cf,ff,f4,4e,75,00,00,00,00
400  data 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
410  data 00,50,0e,0c,0e,0a,16,0c,12,06,08,08,06,08,08,06
420  data 08,08,08,0a,0a,0c,00,00,00,00,00,00,00,00,00,00
430  data *
440  close 1:if s<> 25074 then print "Erreurs dans les DATAs !!!": end
450  print "Ok."
```

2.2 DRIVER D'IMPRIMANTE POUR L'EPSON

L'ATARI ST a l'avantage de comprendre dans son jeu de caractères l'ensemble des caractères ASCII ainsi que presque tous les caractères spéciaux et les accents de la plupart des langues européennes. Mais si vous voulez imprimer ces caractères, la plupart des imprimantes poseront des problèmes.

Une solution serait d'avoir une imprimante avec les codes compatibles IBM, comme par exemple la nouvelle EPSON FX-85 qui peut changer de police de caractères par DIP-switch. Mais si vous possédez une EPSON FX-80 ou bien une des imprimantes qui lui sont compatibles, vous pourrez choisir entre les codes américains et les codes français.

Avec la police de caractères américaine, vous pouvez imprimer les crochets et les accolades auquels vous avez accès sur le ST avec la touche ALT. Pour pouvoir imprimer les accents, il faut choisir la police de caractères française. Le problème vient cependant du fait que les CODES ASCII des accents sur l'ATARI ST et sur l'imprimante EPSON ne correspondent pas.

Pour résoudre ces problèmes, nous avons écrit un petit programme DRIVER qui fera correspondre chaque code ATARI avec la séquence de codes correspondante sur l'imprimante.

Afin que ce programme puisse fonctionner avec tous les programmes qui utilisent la sortie imprimante, il faut modifier les routines TRAP qui servent à l'impression sur imprimante. La méthode pour installer ce programme est semblable à celle du spooler d'imprimante. Après l'initialisation, il faut calculer la place mémoire prise par le programme, qui est résident, et la réserver.

Auparavant, on positionne le vecteur TRAP#13 sur la nouvelle routine et on mémorise l'ancienne valeur.

Dans la routine, on teste si les paramètres sur la pile correspondent à un appel pour la sortie sur imprimante. Si ce n'est pas le cas, on saute à l'ancienne routine TRAP#13. Si l'imprimante était appelée, on envoie la séquence de codes qui, sur la table, correspond au caractère concerné. Si vous voulez étendre la table ou la modifier, afin de la faire correspondre avec une autre imprimante par exemple, faites attention à ce que la dernière donnée de la première table 'chartab' corresponde au premier caractère de la seconde table 'ch1'. La deuxième table est construite de telle façon qu'elle contient d'abord le numéro du caractère moins un, puis le caractère. De cette manière, on peut faire correspondre à chaque caractère de l'ATARI une séquence de codes sur l'imprimante. On pourrait par exemple dessiner un caractère qui n'existe pas sur l'imprimante en le programmant point par point. Votre imagination n'est plus limitée. Vous pouvez également utiliser ce programme avec le spooler d'imprimante.

```
*
*       driver d'imprimante pour imprimante EPSON
*       LE 9/11/85
*

bios    equ     13
keep    equ     $31             garder le programme resident

gemdos  equ     1
setexec equ     5               mise en place du vecteur d'exception
conout  equ     3               impression du caractere
prn     equ     0               No de peripherique de l'imprimante

*                                       calcul de la taille du programme

        move.l  4(sp),a0                adresse de la page de base
        move.l  #$100,d6                taille de la page de base
        add.l   12(a0),d6               plus longueur du texte
        add.l   20(a0),d6               plus longueur de la donnee
        add.l   28(a0),d6               plus longueur bss

*                                       initialisatuion des vecteurs

        move.l  #trap13,-(sp)           nouveau vecteur
        move    #45,-(sp)               No de vecteur
        move    #setexec,-(sp)
        trap    #bios                   positionnement du vecteur
        addq.l  #8,sp
        move.l  d0,trapsve              memorisation de l'ancien vecteur

        clr     -(sp)
        move.l  d6,-(sp)                nombre d'octets
        move    #keep,-(sp)             garder le programme resident
        trap    #gemdos                 retour au bureau
```

```
*                                          nouvelle routine trap#13

trap13   move.l   sp,a2                    memoriser ssp
         btst     #5,(sp)                  appel du superviseur ?
         bne      super                    oui
         move.l   usp,a2                   si non utiliser usp
         subq     #6,a2
super    cmp      #conout,6(a2)            conout-call ?
         bne      normal
         cmp      #prn,8(a2)               imprimante ?
         bne      normal

         move     10(a2),d0                caractere
         lea      chartab(pc),a0           pnt sur table de caracteres speciaux
         moveq    #ch1-chartab-1,d1        longueur de la table
cmploop  cmp.b    (a0)+,d0                 caractere dans la table ?
         beq      printspez                oui, imprimer code EPSON
         dbra     d1,cmploop
         bra      normal                   pas trouve, l'imprimer tel quel

printspez:
         lsl      #2,d1
         lea      adtab(pc),a0             pointeur sur la table d'adresse
         move.l   (a0,d1),a0               pointeur sur la table
         move.b   (a0)+,d0                 longueur de la table moins 1
spzloop  move.b   (a0)+,d1                 caractere lu sur la table
         movem.l  a0/d0,-(sp)             registre sur la table

         move     d1,-(sp)                 caractere
         move     #prn,-(sp)               imprimante
         move     #conout,-(sp)            sortie
         pea      retadr(pc)               adresse retour
         move     sr,-(sp)                 status pour l'appel trap
```

```
normal   move.l   trapsve,a0
         jmp      (a0)                    saut à l'ancien trap 13

retadr   addq.l   #6,sp                   parametre lu sur la pile
         movem.l  (sp)+,a0/d0             reprend le registre
         dbra     d0,spzloop              caractere suivant
         rte

chartab  dc.b     '[\]{|}@~äöü`ÄÖÜï'      caracteres speciaux

ch1      dc.b     0,64                    ï
ch2      dc.b     0,93                    Ü
ch3      dc.b     0,92                    Ö
ch4      dc.b     0,91                    Ä
ch5      dc.b     0,126                   `
ch6      dc.b     0,125                   ü
ch7      dc.b     0,124                   ö
ch8      dc.b     0,123                   ä
ch9      dc.b     6,27,"R",0,126,27,"R",2 ~
ch10     dc.b     6,27,"R",0,64,27,"R",2  @
ch11     dc.b     6,27,"R",0,125,27,"R",2 }
ch12     dc.b     6,27,"R",0,124,27,"R",2 |
ch13     dc.b     6,27,"R",0,123,27,"R",2 {
ch14     dc.b     6,27,"R",0,93,27,"R",2  ]
ch15     dc.b     6,27,"R",0,92,27,"R",2  \
ch16     dc.b     6,27,"R",0,91,27,"R",2  [

         .even

adtab    dc.l     ch1,ch2,ch3,ch4,ch5,ch6
         dc.l     ch7,ch8,ch9,ch10,ch11,ch12
         dc.l     ch13,ch14,ch15,ch16

         .bss
trapsve  ds.l     1                       ancien vecteur TRAP 13
```

```
100  open "R",1,"b:epson.prg",16
110  field#1,16 as bin$
120  a$="": for i=1 to 16: read x$: if x$="*"then 150
130  a=val("&H"+x$): s=s+a:a$=a$+chr$(a): next
140  lset bin$=a$: rec=rec+1: put 1,rec: goto 120
150  data 60,1A,00,00,01,3E,00,00,00,00,00,00,00,04,00,00
160  data 00,00,00,00,00,00,00,00,00,00,00,00,20,6F,00,04
170  data 2C,3C,00,00,01,00,DC,A8,00,0C,DC,A8,00,14,DC,A8
180  data 00,1C,2F,3C,00,00,00,38,3F,3C,00,2D,3F,3C,00,05
190  data 4E,4D,50,8F,23,C0,00,00,01,3E,42,67,2F,06,3F,3C
200  data 00,31,4E,41,24,4F,08,17,00,05,66,04,4E,6A,5D,4A
210  data 0C,6A,00,03,00,06,66,3E,0C,6A,00,00,00,08,66,36
220  data 30,2A,00,0A,41,FA,00,44,72,0F,B0,18,67,06,51,C9
230  data FF,FA,60,22,E5,49,41,FA,00,92,20,70,10,00,10,18
240  data 12,18,48,E7,80,80,3F,01,3F,3C,00,00,3F,3C,00,03
250  data 48,7A,00,0C,40,E7,20,79,00,00,01,3E,4E,D0,5C,8F
260  data 4C,DF,01,01,51,C8,FF,DA,4E,73,5B,5C,5D,7B,7C,7D
270  data 40,7E,84,94,81,9E,8E,99,9A,DD,00,40,00,5D,00,5C
280  data 00,5B,00,7E,00,7D,00,7C,00,7B,06,1B,52,00,7E,1B
290  data 52,02,06,1B,52,00,40,1B,52,02,06,1B,52,00,7D,1B
300  data 52,02,06,1B,52,00,7C,1B,52,02,06,1B,52,00,7B,1B
310  data 52,02,06,1B,52,00,5D,1B,52,02,06,1B,52,00,5C,1B
320  data 52,02,06,1B,52,00,5B,1B,52,02,00,00,00,AE,00,00
330  data 00,B0,00,00,00,B2,00,00,00,B4,00,00,00,B6,00,00
340  data 00,B8,00,00,00,BA,00,00,00,BC,00,00,00,BE,00,00
350  data 00,C6,00,00,00,CE,00,00,00,D6,00,00,00,DE,00,00
360  data 00,E6,00,00,00,EE,00,00,00,F6,00,00,00,18,12,62
370  data 72,04,04,04,04,04,04,04,04,04,04,04,04,04,04,04
380  data *
390  close 1:if s<> 19776 then print "Erreurs dans les DATAs !!!": end
400  print "Ok."
```

2.3. DISQUE VIRTUEL EN RAM POUR ATARI ST

Si vous avez déjà travaillé avec l'assembleur du 68000 ou un compilateur C sur l'ATARI ST, vous savez qu'il y a de nombreuses étapes entre le programme source et le fichier PRG, durant lesquelles plusieurs fichiers sont crées puis détruits. La compilation, l'assemblage, le linkage et la relocation font fonctionner l'unité de disquette et si vous avez déjà écrit un programme, vous savez que ces opérations ne sont pas réalisées en une seule fois : il faut toujours recharger l'éditeur avec le programme source, corriger les erreurs, sauvegarder de nouveau le programme source et recommencer à zéro. Pour de longs programmes, ce processus peut prendre facilement un quart d'heure. Si on cherche à savoir ce qui prend le plus de temps, on se rend vite compte que c'est la sauvegarde et la mémorisation à partir de la disquette. C'est encore plus sensible lorsqu'on travaille avec un seul lecteur car la tête de lecture-écriture est toujours occupée. Mais comment échapper à ce problème?

Une alternative consiste à utiliser un disque dur qui possède un temps d'accès environ 30 fois plus court. Mais il existe une autre possibilité encore plus rapide et qui ne coûte pas un centime : un disque virtuel en RAM!

Que faut-il entendre par là ? C'est très simple : un disque virtuel en RAM est un lecteur de disque qui se trouve dans la mémoire de l'ordinateur. On manipule tout simplement un secteur de la mémoire comme un périphérique, en l'occurrence, comme un lecteur de disque. Si on doit écrire des données sur ce disque, on ne les envoie pas au contrôleur de disque mais dans un secteur particulier de la mémoire. On peut ce permettre ceci grâce à la rapidité du processeur 68000. De plus, on élimine par là tout le temps de recherche et de positionnement de la tête, nécessaire avec un lecteur de disquette normal.

La lecture et l'écriture sur un disque virtuel se réduit à une copie de donnée d'un secteur de mémoire (le disque virtuel),

dans un autre (le tampon de disquette). Le programme qui installe un disque RAM doit donc se charger de ce transfert de données. Outre cette petite partie, il faut assurer la liaison avec le système d'exploitation. Mais ATARI y a déjà pensé. Il suffit de dévier trois vecteurs. Ces vecteurs servent au branchement d'un disque dur, mais ils conviennent parfaitement à notre application. Les trois vecteurs en question concernent les routines du BIOS pour la lecture et l'écriture des secteurs, la mémorisation des paramètres de blocs (qui donne des information sur la taille et l'organisation d'une disquette) ainsi que la fonction de media change, qui indique au DOS quand une disquette a été changée (difficile avec un lecteur virtuel, n'est-ce pas ?). Notre lecteur virtuel en RAM doit avoir le numéro de lecteur C, en BIOS la valeur 2 est utilisée (0=lecteur A, 1=lecteur B). Dans le programme, les vecteurs doivent être positionnées sur notre routine qui teste si le lecteur C est concerné. Cela est réalisé en testant le numéro du lecteur inscrit comme paramètre sur la pile. Si le lecteur C n'est pas concerné, il suffit de se brancher sur les routines qui manipulent les lecteurs A et B. Pour que le programme soit agréable à utiliser, la taille du disque virtuel sera donnée sous forme de paramètre. On précise ce paramètre en Koctets comme pour le spooler d'imprimante (les deux programmes peuvent fonctionner ensemble). On n'est pas limité à la taille habituelle des disquettes (180, 360 ou 720K) mais, selon les besoins, on peut donner une valeur de 80 Koctets (ou moins si cela vous semble intéressant) à 640 koctets sur un ATARI ST+. Sur une machine de 512 Koctets, on peut avoir un disque virtuel faisant de 100 à 120 Koctets environ. (Cela dépend naturellement de la taille de votre programme).

Si on ne donne pas de paramètre à ce programme, la valeur est fixée automatiquement à 100 Koctets. Vous pouvez bien sûr adapter cette valeur directement sur le listing en fonction de vos besoins. Si vous vous servez du chargeur BASIC, il vous faudra adapter les deux octets soulignés à la nouvelle valeur (format : octet de poids fort, octet de poids faible; soit 01, 2C pour 300 Ko). Evidemment, vous ne devrez pas oublier de changer la somme-test en conséquence.

Le programme configure automatiquement le paramètre de bloc du BIOS à partir de la capacité de la disquette. Il crée un secteur de chargement dans le disque virtuel et efface le catalogue. Lorsque le programme est arrêté, la place utilisée est calculée automatiquement, comme pour le spooler, et elle est réservée.

Comment installer le disque virtuel?

Exécutez le programme (avec ou sans donner de paramètre). Pour cela, cliquez avec la souris sur un icône de disquette (lecteur A ou B) et choisissez dans les options du desktop le point 'Installer une unité disque...'. Donnez au lecteur le caractère de reconnaissance 'C' pour la description 'DISQUE RAM' puis cliquez sur 'installer'. Sur le desktop apparaît alors un nouvel icône de disque qui porte les signes 'C' et 'DISQUE RAM'. Vous pouvez ouvrir le disque virtuel en cliquant deux fois sur ce symbole. Une fenêtre apparaît : son nom est 'C' et elle contient 0 octet utilisé par 0 objet. Vous pouvez maintenant copier des programmes ou des données du lecteur A ou B sur le disque virtuel.

Cela se passe exactement comme s'il s'agissait d'un lecteur normal. Essayez maintenant de charger un programme à partir du disque virtuel. Ca marche ! Les programmes de 100K mettent moins d'une seconde à être chargés.
Quelle est la meilleure utilisation du disque virtuel ?
Si vous écrivez beaucoup de programmes ou de texte, ou que vous assemblez des programmes, vous pouvez y inscrire votre éditeur, le programme source et s'il vous reste de la place, le compilateur, l'assembleur et le linker. Cela vous permettra de travailler plus rapidement.

Un assemblage qui dure 10 minutes sur un lecteur normal prend moins d'une minute avec le disque virtuel. C'est un gain de temps qui raccourcira sensiblement la phase de développement. Faites attention : n'oubliez pas que les données sur le disque virtuel se trouvent en RAM et qu'elles seront perdues si vous éteignez l'ordinateur.

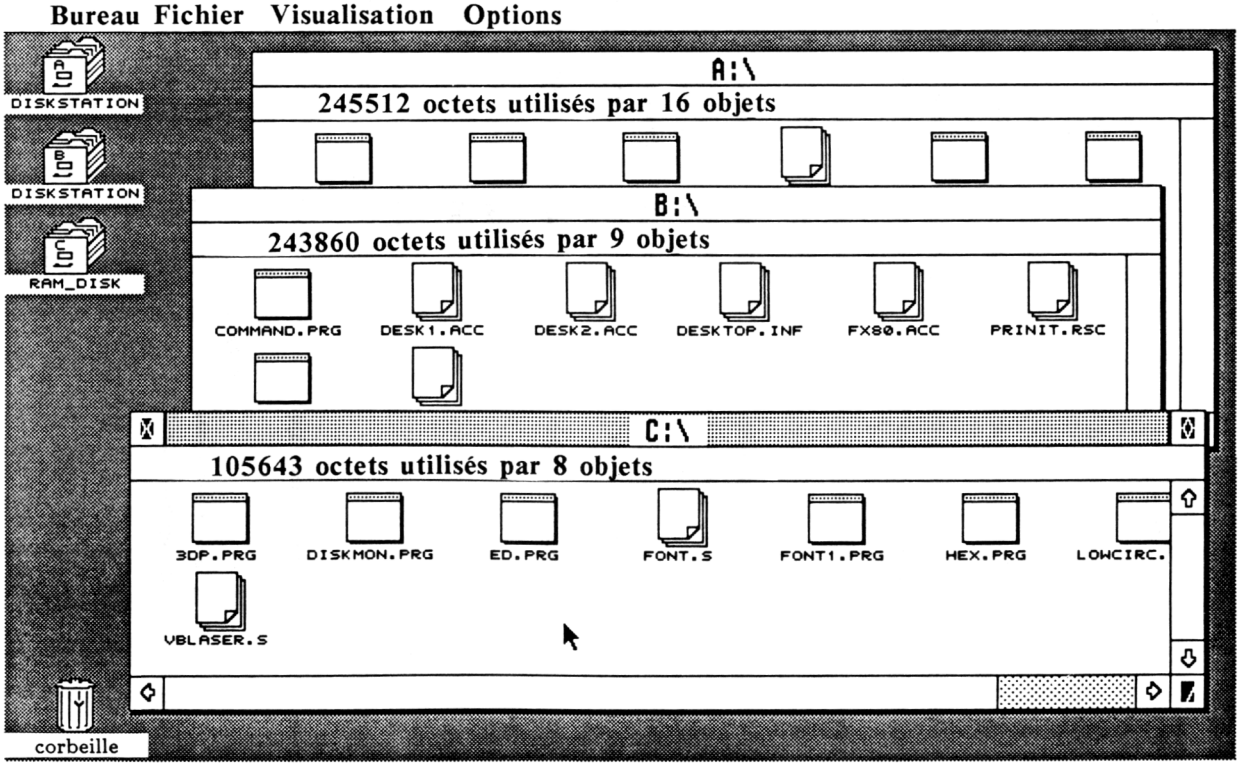
Copiez donc le contenu du disque virtuel sur une vraie disquette avant de tout éteindre! Nous vous conseillons de faire la même chose si vous écrivez des programmes risquant de se planter.

Encore quelques remarques sur la manipulation du disque virtuel. Il n'est pas possible de faire une copie à partir d'un lecteur normal sur le disque virtuel.

Suivez la méthode suivante pour faire une copie : ouvrez une fenêtre avec le lecteur sur lequel vous voulez copier et tirez l'icône de la disquette qui doit être copiée sur cette fenêtre. N'essayez pas non plus de formatter le disque virtuel, cela pourrait endommager les disquettes du lecteur A ou B. Pour effectuer l'opération similaire, prenez les icônes des fichiers à supprimer sur le disque virtuel et tirez les dans la corbeille. Cela se fait très rapidement avec le disque virtuel.

L'installation du disque virtuel après la mise en route de l'ordinateur peut être faite automatiquement par l'ATARI. Ecrivez sur votre disquette système un fichier dont le nom est 'AUTO' et copiez-y le programme d'installation 'RAMDISC.PRG'. Si vous choisissez l'option 'sauvegarde' sur le desktop, votre configuration avec le disque RAM sera sauvegardée dans le fichier 'DESKTOP.INF'. A chaque mise en route, le disque virtuel sera automatiquement installé et une fenêtre sera directement ouverte pour le lecteur 'C'.

Bureau Fichier Visualisation Options
DISKSTATION
DISKSTATION
INSTALLER UNE UNITE DISQUE
Nom de l'unité : C
Nom de l'icône : RAM_DISK|____
|INSTALLER| |SUPPRIMER| |ANNULER|
corbeille


Bureau Fichier Visualisation Options
DISKSTATION
DISKSTATION
RAM_DISK
corbeille
A:\
245512 octets utilisés par 16 objets
B:\
243860 octets utilisés par 9 objets
COMMAND.PRG
DESK1.ACC
DESK2.ACC
DESKTOP.INF
FX80.ACC
PRINIT.RSC
C:\
105643 octets utilisés par 8 objets
3DP.PRG
DISKMON.PRG
ED.PRG
FONT.S
FONT1.PRG
HEX.PRG
LOWCIRC.
VBLASER.S

```
*
*       RAM disc pour ATARI ST
*       LE/RB, 6/11/85
*

hdv_bpb equ         $472                parametre de bloc pour la bios
hdv_rw  equ         $476                lecture/ecriture de secteurs
hdv_mediach         equ         $47e

drvbits equ         $4c2                bit-vecteur d'activation du lecteur

gemdos  equ         1
keep    equ         $31

xbios   equ         14
super   equ         38

default equ         100                 capacite standard en ko

init    move.l      4(sp),a0            base page address
        move.l      #$100,d6            taille de la base page
        add.l       12(a0),d6           longueur du texte
        add.l       20(a0),d6           longueur des donnees
        add.l       28(a0),d6           longueur bss

        moveq       #0,d7
        moveq       #0,d0
        lea         129(a0),a0          pointeur sur ligne de commande
nextchr move.b      (a0)+,d0            premier caractere de la ligne de commande
        sub.b       #'0',d0
        bmi         exit
        cmp.b       #9,d0               chiffre?
        bgt         exit
        mulu        #10,d7
        add         d0,d7               prochain chiffre
        bra         nextchr
exit    tst         d7                  saisie effectuee  ?
```

```
        bne     ok
        move.w  #default,d7         valeur par defaut

ok      moveq   #0,d1
        move    d7,d1               capacite en ko
        add     #9,d1               plus 9 k
        lsl.l   #8,d1
        lsl.l   #2,d1               * 1024
        add.l   d1,d6               additionner à besoin en memoire


        move.l  #init1,-(sp)
        move    #super,-(sp)        initialisation en mode superviseur
        trap    #xbios
        addq.l  #6,sp

        clr     -(sp)
        move.l  d6,-(sp)            nombre d'octets
        move    #keep,-(sp)         laisser le programme resident
        trap    #gemdos             retour au desktop


init1   move.l  hdv_bpb,bpbsave
        move.l  #bpb,hdv_bpb

        move.l  hdv_rw,rwsave       vecteurs pointent sur nouvelle routine
        move.l  #rw,hdv_rw

        move.l  hdv_mediach,mediasave
        move.l  #media,hdv_mediach


install moveq   #0,d1
        lea     ramdisk,a0
        move    #2*9*512/4-1,d0     effacer pistes 0 et 1
iloop1  move.l  d1,(a0)+            du ram disc
        dbra    d0,iloop1
```

```
*                                        generer un boot secteur
        lea     ramdisk+11,a0
        lea     boottab,a1
        moveq   #tabend-boottab-1,d0
bloop   move.b  (a1)+,(a0)+      copier les donnees dans le boot secteur
        dbra    d0,bloop

        move    d7,numcl         capacite en ko dans bpb

        lsl     #1,d7            capacite en secteurs
        add     #18,d7           plus 18 secteurs
        lea     ramdisk+19,a0
        move.b  d7,(a0)+         lo byte
        lsr     #8,d7
        move.b  d7,(a0)          hi byte

        or.l    #%100,drvbits    appel drive c
        rts

bpb:    cmp     #2,4(sp)         drive c ?
        beq     bpb1             oui

        move.l  bpbsave,a0       ancienne routine
        jmp     (a0)

bpb1    move.l  #bpbtab,d0       pointeur sur bios parameter block
        rts
```

```
rw      cmp     #2,14(sp)       drive c ?
        beq     rw1             oui

        move.l  rwsave,a0       ancienne routine
        jmp     (a0)

rw1     move    12(sp),d0       recno, numero de secteur logique
        ext.l   d0
        lsl.l   #8,d0
        lsl.l   #1,d0           fois 512

        move.l  6(sp),a0        adresse tampon

        move    10(sp),d1       nombre de secteurs
        subq    #1,d1
        lea     ramdisk,a1      adresse de base
        add.l   d0,a1           plus adresse relative dans ram disc

        move    4(sp),d0        rwflag
        btst    #0,d0           lecture ?
        beq     rloop0          oui
        exg     a0,a1           echanger objet et source

rloop0  move    #511,d0         un secteur
rloop   move.b  (a1)+,(a0)+     copier tampon
        dbra    d0,rloop
        dbra    d1,rloop0       secteur suivant
        moveq   #0,d0           ok
        rts


media   cmp     #2,4(sp)        drive c ?
        beq     media1          oui

        move.l  mediasave,a0    ancienne routine
        jmp     (a0)
```

```
media1  moveq   #0,d0           disquette pas changee
        rts

        .data
bpbtab:
recsiz: dc.w    $200            taille du secteur
clsiz   dc.w    2               taille du cluster en secteurs
clsizb  dc.w    $400            taille du cluster (groupe) en octets
rdlen   dc.w    7               taille du catalogue en secteurs
fsiz    dc.w    5               taille du fat
fatrec  dc.w    6               secteurs du fat
datrec  dc.w    18              secteurs pour la gestion
numcl   ds.w    1               capacite en ko
flags   ds.w    8

boottab:                *       donnees en format 8086
        dc.b    0,2             octets par secteur
        dc.b    2               secteurs par cluster
        dc.b    1,0             secteurs reserves
        dc.b    2               fats
        dc.b    112,0           directory entries
        ds.b    2               sectors on media
        dc.b    0               media descriptor
        dc.b    5,0             sectors per fat
        dc.b    9,0             sectors per track
        dc.b    1,0             sides
        dc.b    0               hidden
tabend  equ     *

        .bss

bpbsave ds.l    1               place pour anciens vecteurs-floppy
rwsave  ds.l    1
mediasave       ds.l    1

ramdisk equ     *               le ram disc commence ici
```

```
100  open "R",1,"b:ramdisc.prg",16
110  field#1,16 as bin$
120  a$="": for i=1 to 16: read x$: if x$="*"then 150
130  a=val("&H"+x$): s=s+a:a$=a$+chr$(a): next
140  lset bin$=a$: rec=rec+1: put 1,rec: goto 120
150  data 60,1A,00,00,01,5E,00,00,00,32,00,00,00,0C,00,00
160  data 00,00,00,00,00,00,00,00,00,00,00,00,20,6F,00,04
170  data 2C,3C,00,00,01,00,DC,A8,00,0C,DC,A8,00,14,DC,A8
180  data 00,1C,7E,00,70,00,41,E8,00,81,10,18,90,3C,00,30
190  data 6B,0E,B0,3C,00,09,6E,08,CE,FC,00,0A,DE,40,60,EA
200  data 4A,47,66,04,3E,3C,00,64,72,00,32,07,D2,7C,00,09
210  data E1,89,E5,89,DC,81,2F,3C,00,00,00,62,3F,3C,00,26
220  data 4E,4E,5C,8F,42,67,2F,06,3F,3C,00,31,4E,41,23,F9
230  data 00,00,04,72,00,00,01,90,23,FC,00,00,00,E8,00,00
240  data 04,72,23,F9,00,00,04,76,00,00,01,94,23,FC,00,00
250  data 01,00,00,00,04,76,23,F9,00,00,04,7E,00,00,01,98
260  data 23,FC,00,00,01,4A,00,00,04,7E,72,00,41,F9,00,00
270  data 01,9C,30,3C,08,FF,20,C1,51,C8,FF,FC,41,F9,00,00
280  data 01,A7,43,F9,00,00,01,7E,70,11,10,D9,51,C8,FF,FC
290  data 33,C7,00,00,01,6C,E3,4F,DE,7C,00,12,41,F9,00,00
300  data 01,AF,10,C7,E1,4F,10,87,00,B9,00,00,00,04,00,00
310  data 04,C2,4E,75,0C,6F,00,02,00,04,67,08,20,79,00,00
320  data 01,90,4E,D0,20,3C,00,00,01,5E,4E,75,0C,6F,00,02
330  data 00,0E,67,08,20,79,00,00,01,94,4E,D0,30,2F,00,0C
340  data 48,C0,E1,88,E3,88,20,6F,00,06,32,2F,00,0A,53,41
350  data 43,F9,00,00,01,9C,D3,C0,30,2F,00,04,08,00,00,00
360  data 67,02,C1,49,30,3C,01,FF,10,D9,51,C8,FF,FC,51,C9
370  data FF,F4,70,00,4E,75,0C,6F,00,02,00,04,67,08,20,79
380  data 00,00,01,98,4E,D0,70,00,4E,75,02,00,00,02,04,00
390  data 00,07,00,05,00,06,00,12,00,00,00,00,00,00,00,00
400  data 00,00,00,00,00,00,00,00,00,00,00,02,02,01,00,02
410  data 70,00,00,00,00,05,00,09,00,01,00,00,00,00,00,4C
420  data 1C,06,0E,06,0E,06,0C,10,06,0E,0C,20,08,10,1C,2E
430  data *
440  close 1:if s<> 26687 then print "Erreur dans les DATAs !!!": end
450  print "Ok."
```

2.4. SPOOLER D'IMPRIMANTE POUR L'ATARI ST

Dans cette section, nous allons vous présenter un outil utile qui peut vous faire gagner du temps. S'il vous est déjà arrivé de rester devant votre ordinateur en attendant qu'un listing de 10 pages soit imprimé, alors vous comprenez de quoi nous parlons.

La mécanique d'une imprimante ne suit pas la rapidité de transmission des données de l'ordinateur et il faut constamment attendre qu'elle soit prête.

C'est pour cette raison que de nombreuses imprimantes ont un tampon dans lequel les données sont inscrites avant d'être imprimées. Les données sont ensuite transférées à la vitesse d'impression de l'imprimante.

La taille courante du tampon est 2Koctets, ce qui correspond environ à une page au format A4. Si le texte à imprimer est plus grand, le tampon est vite rempli par le transfert et l'ordinateur doit attendre de nouveau. Une solution serait d'avoir une imprimante avec un tampon plus grand. Mais de tels appareils sont beaucoup plus chers. Or nous avons un ordinateur avec 512 Koctets (ou 1 mégaoctets!) Il est plus simple de mettre le tampon dans l'ordinateur!

Afin de comprendre le mode de fonctionnement du programme qui suit, nous devons d'abord expliquer rapidement comment s'effectue le transfert de données de l'ordinateur vers l'imprimante. Les données sont envoyées en parallèle sur une sortie dite 'centronics' qui transfère 8 bits à la fois, soit un octet, par huits conducteurs. Afin que l'ordinateur et l'imprimante soient coordonnés au moment du transfert, deux conducteurs dits de 'handshake' sont nécessaires. Lorsque l'imprimante est prête à recevoir un octet, elle envoie un signal à l'état bas sur le conducteur 'busy'.
L'ordinateur renvoie alors à l'imprimante un signal bas sur le conducteur 'strobe'.

Si nous mettons un tampon intermédiaire dans l'ordinateur pour les données à envoyer à l'imprimante, nous avons besoin de deux routines. La première qui remplace la sortie imprimante précédente écrit le caractère dans le tampon. La seconde sert à envoyer un caractère vers l'imprimante à chaque fois qu'elle est prête.

Le programme est fait de telle façon qu'il peut gérer des tampons faisant jusqu'à 63 Koctets. Si vous démarrez le programme en cliquant avec la souris, 32 Koctets sont réservés comme tampon d'imprimante. Cela permet de mémoriser environ 15 pages de texte. Si vous installez le programme pour qu'il soit paramètrable avec le TOS, vous pourrez fixer la taille du tampon à la mise en route. Une valeur de 1 à 63 indique la taille en Koctets. Le programme réserve automatiquement la place à partir du système d'exploitation. Cette mémoire n'est bien sûr pas utilisable par d'autres programmes. S'il ne reste plus assez de place pour certains programmes (Ce ne sera sans doute le cas que pour les machines de 512 K ou moins), vous devrez réduire la taille du tampon.

Si vous utilisez souvent le spooler et que vous vouliez l'installer à chaque mise en route du st, le système d'exploitation dispose d'une solution très pratique. Créez un fichier dont le nom est 'AUTO' sur la disquette système et copiez y le programme. A la mise en route, les programmes de cette section seront mis en route automatiquement. Venons en à une courte description du programme.

Afin que le programme puisse réserver la place pour lui et pour le tampon, on calcule tout d'abord sa taille. Cette donnée peut se trouver dans la page de base qui fait $100 octets et qui se trouve immédiatement avant le programme.

L'adresse de la page de base est prise sur la pile. On additionne alors la longueur du texte, des données et du segment de mémoire avec la longueur de la page de base. Dans la page de base se trouve également la ligne de commande, c'est à dire le texte qui sert de paramètre à notre programme, soit la longueur du tampon en Koctets. Il faut lire les chiffres et les transformer en nombres binaires.

Si aucun paramètre n'a été donné, nous prenons la valeur 32 par défaut. En effectuant une rotation à gauche de 10 bits, nous obtenons la valeur en octets. Nous inscrivons cette valeur dans la table de données du tampon comme étant la taille du tampon. Cette table est construite de la même manière que le 'iorec' du système d'exploitation (voir 'Anatomie de l'ATARI'). Puis nous branchons le vecteur TRAP#13 sur notre propre routine dans laquelle nous testons si nous avons affaire à une sortie sur imprimante ou bien au statut de l'imprimante. Cela est effectué en testant les paramètres sur la pile. Si une autre routine était appelée, nous nous branchons de nouveau sur la routine TRAP#13.

S'il s'agit de l'impression d'un caractère, plusieurs possibilités se présentent. Dans le cas où le tampon est encore libre, on essaie d'envoyer directement le caractère sur l'imprimante. Si l'imprimante n'est pas prête, on écrit le caractère dans le tampon. On fait de même si le tampon contient déjà des caractères. Si le tampon est rempli, on attend au plus 30 secondes que de la place soit faite. Quand le temps est écoulé, on signale à la routine que le caractère n'a pu être envoyé. Cela ne peut arriver que lorsque le tampon est plein (32Koctets) et que l'imprimante ne peut prendre aucun autre caractère. Mais comment les caractères sont-ils transférés du tampon vers l'imprimante?

Nous utilisons pour cela les possibilités d'interruption du connecteur BUSY de l'imprimante. Cette interruption est mise en place dans la routine d'initialisation et elle est branchée sur notre routine 'BUSYINT'. Chaque fois que l'imprimante est prête à recevoir un autre caractère, elle envoie une interruption à l'ordinateur. La routine d'interruption teste s'il reste encore des caractères dans le tampon. Dans ce cas, on prend un caractère dans le tampon et on l'envoie vers l'imprimante. La routine d'interruption est alors terminée et on peut retourner au programme qui a été arrêté. l'avantage de ce processus d'interruption est que l'ordinateur n'attend jamais l'imprimante (sauf quand le tampon est plein, mais il contient alors déjà 15 pages).

Si vous envoyez un texte de 10 pages vers l'imprimante avec le spooler, l'ordinateur est de nouveau disponible après quelques secondes, alors que l'imprimante en a encore pour plusieurs minutes.
Vous trouverez ci-dessous le listing en assembleur du programme ainsi qu'un chargeur BASIC qui crée un fichier programme sur disquette.

```
*
*        spooler d'imprimante pour atari st
*
*      LE/RB, 5/11/85
*

bios      equ       13
keep      equ       $31          garder le programme resident

gemdos    equ       1
setexec   equ       5            mettre en place les vecteurs d'exception
conout    equ       3            affiche caractere
constat   equ       8            statut de l'affichage
prn       equ       0            device # de l'imprimante
savptr    equ       $4a2         save area pour registre
hz_200    equ       $4ba         200 hz system takt

xbios     equ       14
mfpint    equ       13                 installation mfp interrupt

mfp       equ       $fffa01            mfp 68901
psg       equ       $ff8800            psg ym 2149
isrb      equ       $10                interrupt service register b

default   equ       32                 taille standard du tampon en ko
timeout   equ       30                 30 secondes timeout

*                                      calcul de la taille du programme
          move.l    4(sp),a0           base page address
```

```
move.l          #$100,d6        taille de la base page
        add.l   12(a0),d6       plus longueur du texte
        add.l   20(a0),d6       plus longueur des donnees
        add.l   28(a0),d6       plus longueur bss

*                               taille tampon dans ligne de commande
        moveq   #0,d7
moveq           #0,d0
        lea     129(a0),a0      pointeur sur ligne de commande
nextchr move.b  (a0)+,d0        lire caractere
        sub.b   #'0',d0
        bmi     exit            aucun chiffre
        cmp.b   #9,d0
        bgt     exit            aucun chiffre
        mulu    #10,d7          prochain emplacement
        add     d0,d7
        bra     nextchr

exit    tst     d7              saisir nombre ?
        bne     ok              oui
        move    #default,d7     sinon prendre valeur par defaut
ok      ext.l   d7
        moveq   #10,d0
        lsl.l   d0,d7           calculer valeur en octets
        add.l   d7,d6           additionner à place nécessaire
        move    d7,laenge       et mettre dans iorec

*                               initialiser les vecteurs

        move.l  #trap13,-(sp)   nouveau vecteur
        move    #45,-(sp)       numéro de vecteur
        move    #setexec,-(sp)
        trap    #bios           mettre en place vecteur
        addq.l  #8,sp
        move.l  d0,trapsve      mémoriser ancien vecteur

        move.l  #busyint,-(sp)
```

```
move                #0,-(sp)            int nummer
        move        #mfpint,-(sp)
        trap        #xbios              centronics interrupt enable
        addq.l      #8,sp

        clr         -(sp)
        move.l      d6,-(sp)            2nombre d'octets
move                #keep,-(sp)         garder le programme résident
        trap        #gemdos             retour au desktop

*                                       nouvelle routine trap#13
trap13  move.l      sp,a2               mémoriser ssp
        btst        #5,(sp)             appel du  superviseur ?
        bne         super               oui
        move.l      usp,a2              sinon utiliser usp
        subq        #6,a2
super   cmp         #conout,6(a2)       conout-call ?
        bne         normal
        cmp         #prn,8(a2)          printer ?
        bne         normal

        move.l      savptr,a1           pointeur sur save area
        move        (sp)+,-(a1)         sauvegarder statut
        move.l      (sp)+,-(a1)         return adress
        move.l      a1,savptr           save ptr updata

        move        10(a2),d1           character
        bsr         print

        move.l      savptr,a1
        move.l      (a1)+,-(sp)         return adress
        move        (a1)+,-(sp)         statut
        move.l      a1,savptr
        rte

normal:
        cmp         #constat,6(a2)      statut imprimante?
```

```
        bne     norm1
        cmp     #prn,8(a2)
        bne     norm1                   sur ancien vecteur trap#13

        moveq   #-1,d0                  mettre statut ok
        bsr     getptr                  prendre pointeur
        move    tail(a0),d2
bsr             wrap
        cmp     head(a0),d2             de la place dans le tampon?
        bne     platz                   oui
        moveq   #0,d0                   busy, pas de place
platz   rte

norm1   move.l  trapsve,a0              vers ancien trap#13
        jmp     (a0)

print   move    #$2700,sr               inhiber interruption
        bsr     getptr                  pointeur sur iorec et mfp
        move    head(a0),d2
        cmp     tail(a0),d2             tampon vide ?
        bne     inbuff                  non, caractere dans tampon

loop    btst    #0,(a1)                 imprimante busy ?
        bne     inbuff                  oui, dans tampon

notbusy lea     psg,a2                  adresse du psg
        move.b  #15,(a2)                numero de registre du port b
        move.b  d1,2(a2)                envoyer octets

        move.b  #14,(a2)                numero de registre du port a
        move.b  (a2),d0
        and.b   #$df,d0                 strobe low
        move.b  d0,2(a2)

        or.b    #$20,d0                 strobe high
        move.b  d0,2(a2)
```

```
        moveq   #-1,d0                  ok
        rts
inbuff  move    tail(a0),d2             écrit pointeur
        bsr     wrap                    augmenter
        cmp     head(a0),d2             tampon plein?
beq             buffull                 oui
inbuff1 move.l  (a0),a1                 adresse du tampon
        move.b  d1,(a1,d2)              écrire caractères dans tampon
        move    d2,tail(a0)             mémoriser indice de fin
        moveq   #-1,d0                  caractère envoyé
        rts

buffull move.l  hz_200,d0
        add.l   #timeout*200,d0         nombre de secondes d'attente
        move    #$2300,sr               activer interruption
wait    cmp     head(a0),d2             encore de la place dans le tampon?
        bne     inbuff1                 oui, caractère dans tampon
        cmp.l   hz_200,d0               temps écoulé?
        bhi     wait                    non, continue attente

        moveq   #0,d0                   caractère pas envoyé
        rts

*       interrupt routine pour envoyer un caractère vers l'imprimante

busyint movem.l d0-d2/a0-a2,-(sp)       sauvegarder registre
        bsr     getptr                  prendre pointeur
        move    head(a0),d2
        cmp     tail(a0),d2             tampon d'expédition vide?
        beq     leer                    oui, déjà fini
        bsr     wrap                    augmenter pointeur de lecture
        move.l  (a0),a2                 adresse tampon
        move.b  (a2,d2),d1              lire caractère dans tampon et
        bsr     notbusy                 envoyer vers imprimante
        move    d2,head(a0)             mémorise nouvel indice de début
```

```
leer    bclr    #0,isrb(a1)          effacer bit d'interruption
        movem.l (4sp)+,d0-d2/a0-a2   relire registre
        rte

getptr  lea     iorec,a0             pointeur sur tampon de données
        lea     mfp,a1
        rts

wrap    addq    #1,d2                pointeur sur position suivante
        cmp     len(a0),d2           fin du tampon ?
        bcs     nowrap               non
        moveq   #0,d2                sinon, recommencer du début
nowrap  rts

        .data
iorec   dc.l    buf                  adresse du tampon
laenge  ds.w    1                    taille du tampon
        dc.w    0                    écrire indice
        dc.w    0                    lire indice

buffer  equ     0                    offset dans iorec
len     equ     4
head    equ     6
tail    equ     8

        .bss
trapsve ds.l    1                    ancien vecteur trap#13
buf     equ     *                    debut de la mémoire tampon
```

```
100     open "R",1,"b:spool.prg",16
110     field#1,16 as bin$:a$="":for i=1 to 16:read x$:if x$="*"then 150
130     a=val("&H"+x$):s=s+a:a$=a$+chr$(a):next
140     lset bin$=a$:rec=rec+1:put 1,rec: goto 120
150     data 60,1A,00,00,01,B0,00,00,00,0A,00,00,00,04,00,00
160     data 00,00,00,00,00,00,00,00,00,00,00,00,20,6F,00,04
170     data 2C,3C,00,00,01,00,DC,A8,00,0C,DC,A8,00,14,DC,A8
180     data 00,1C,7E,00,70,00,41,E8,00,81,10,18,90,3C,00,30
190     data 6B,0E,B0,3C,00,09,6E,08,CE,FC,00,0A,DE,40,60,EA
200     data 4A,47,66,02,7E,20,48,C7,70,0A,E1,AF,DC,87,33,C7
210     data 00,00,01,B4,2F,3C,00,00,00,7C,3F,3C,00,2D,3F,3C
220     data 00,05,4E,4D,50,8F,23,C0,00,00,01,BA,2F,3C,00,00
230     data 01,6C,3F,3C,00,00,3F,3C,00,0D,4E,4E,50,8F,42,67
240     data 2F,06,3F,3C,00,31,4E,41,24,4F,08,17,00,05,66,04
250     data 4E,6A,5D,4A,0C,6A,00,03,00,06,66,30,0C,6A,00,00
260     data 00,08,66,28,22,79,00,00,04,A2,33,1F,23,1F,23,C9
270     data 00,00,04,A2,32,2A,00,0A,61,42,22,79,00,00,04,A2
280     data 2F,19,3F,19,23,C9,00,00,04,A2,4E,73,0C,6A,00,08
290     data 00,06,66,20,0C,6A,00,00,00,08,66,18,70,FF,61,00
300     data 00,C2,34,28,00,08,61,00,00,C8,B4,68,00,06,66,02
310     data 70,00,4E,73,20,79,00,00,01,BA,4E,D0,46,FC,27,00
320     data 61,00,00,A0,34,28,00,06,B4,68,00,08,66,2E,08,11
330     data 00,00,66,28,45,F9,00,FF,88,00,14,BC,00,0F,15,41
340     data 00,02,14,BC,00,0E,10,12,C0,3C,00,DF,15,40,00,02
350     data 80,3C,00,20,15,40,00,02,70,FF,4E,75,34,28,00,08
360     data 61,6E,B4,68,00,06,67,0E,22,50,13,81,20,00,31,42
370     data 00,08,70,FF,4E,75,20,39,00,00,04,BA,D0,BC,00,00
380     data 17,70,46,FC,23,00,B4,68,00,06,66,DC,B0,B9,00,00
390     data 04,BA,62,F2,70,00,4E,75,48,E7,E0,E0,61,24,34,28
400     data 00,06,B4,68,00,08,67,0E,61,26,24,50,12,32,20,00
410     data 61,82,31,42,00,06,08,A9,00,00,00,10,4C,DF,07,07
420     data 4E,73,41,F9,00,00,01,B0,43,F9,00,FF,FA,01,4E,75
430     data 52,42,B4,68,00,04,65,02,74,00,4E,75,00,00,01,BE
440     data 00,00,00,00,00,00,00,00,00,44,06,12,06,88,AE,18
450     data 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,*
460     close 1:if s<> 29742 then print "Erreur dans les DATAs !!!": end
470     print "Ok."
```

2.5. MISE EN ROUTE AUTOMATIQUE DES APPLICATIONS DU TOS

Le système d'exploitation de l'ATARI ST est chargé directement à partir de la disquette et exécuté en RAM. Pour réaliser cela, l'ordinateur possède une 'boot-ROM'. C'est là que s'effectue la mise en place des programmes dans l'ordinateur à la mise sous tension. Le rôle de la boot-ROM est de charger un secteur de la disquette et d'exécuter le programme situé sur ce secteur. Ce programme charge le système d'exploitation puis GEM.
Le secteur de chargement est situé sur le premier secteur de la disquette (secteur 1, piste 0) et contient outre le programme de chargement, des données sur le format de la disquette, telles la capacité, le nombre de pistes et de secteurs ainsi que la taille et l'ordre du catalogue. Le programme de chargement en question ne se trouve que sur la disquette système. Afin que l'ordinateur puisse le reconnaître, la somme des mots du secteur (256 mots= 512 octets) est de $1234 sur une disquette système.

Normalement, le GEM desktop, c'est à dire la partie utilisateur de l'ATARI ST, est exécuté après le chargement du système d'exploitation. Mais le système d'exploitation prévoie la possibilité de charger et d'exécuter non le desktop mais un programme dont le nom est COMMAND.PRG. Il peut s'agir par exemple d'un programme qui fonctionne sous TOS. Comment obtenir cela avec le système d'exploitation?

C'est le secteur de chargement qui détermine cette possibilité. Si le secteur de chargement contient la valeur 0 à un endroit défini, le desktop est chargé normalement. Mais si la valeur de cette adresse est différente de zéro, le programme COMMAND.PRG est chargé. La valeur de cette adresse est placée dans la variable d'exploitation 'cmdload' ($482). Quand le système d'exploitation est chargé, cette variable détermine s'il faut ou non charger le programme COMMAND.PRG. Si nous voulons exécuter un logiciel d'application à la place du desktop, il suffit de modifier le secteur de chargement en conséquence.

Nous avons écrit un petit programme permettant cela. Le programme est divisé en quatre parties : d'abord, le secteur de chargement est chargé à partir du lecteur A, puis le label pour 'cmdload' est mis en place dans le secteur de chargement. Nous pourrions alors sauvegarder le secteur de chargement. Mais il faut se rappeler que la somme des mots du secteur de chargement doit rester la même. Or elle est modifiée par l'octet changé et le système d'exploitation ne reconnaitrait plus la disquette comme étant une disquette système. Au lieu de recalculer la somme, nous utilisons une fonction du système d'exploitation. La fonction 'protobt' sert à créer ou bien à modifier un secteur de chargement. Lorsqu'on fixe le paramètre, il faut que le secteur de chargement soit toujours utilisable. Il ne faut donc pas modifier les autres paramètres. Avec la fonction en question, la somme est recalculée et elle est inscrite dans le secteur de chargement. Quand cela est fait, le secteur peut être recopié.

Maintenant, nous pouvons copier un programme d'application sous le nom 'COMMAND.PRG'. Il sera exécuté automatiquement par le système d'exploitation à la place du desktop.

```
*
*          modification du secteur de chargement pour cmdload
*          LE 11/11/85
*

gemdos     equ       1

xbios      equ       14

floprd     equ       8          lire secteur
flopwr     equ       9          ecrire secteur
protobt    equ       18         generer boot secteur

cmdload    equ       $1e        debut dans le secteur de chargement

*                    charger le boot secteur

           move      #1,-(sp)             un secteur
           move      #0,-(sp)             face zero
           move      #0,-(sp)             piste zero
           move      #1,-(sp)             secteur zero
           move      #0,-(sp)             lecteur A
           clr.l     -(sp)
           move.l    #puffer,-(sp)        adresse du tampon
           move      #floprd,-(sp)        lire boot secteur
           trap      #xbios
           add.l     #20,sp

           tst       d0                   y-a t-il eu une erreur ?
           bne       exit                 oui, arreter le programme

*                    modifier boot secteur

           lea       puffer,a0            adresse du tampon
           move.b    #1,cmdload(a0)       mise en place du label pour cmdload
```

```
*                       rendre le boot secteur reutilisable

        move    #1,-(sp)        rendre le boot secteur utilisable
        move    #-1,-(sp)       ne pas changer le type du disque
        move.l  #-1,-(sp)       ne pas modifier le numero de serie
        move.l  #puffer,-(sp)   adresse du boot secteur
        move    #protobt,-(sp)  appel de la fonction
        trap    #xbios
        add.l   #14,sp

*                       re-ecrire le boot secteur modifie

        move    #1,-(sp)        un secteur
        move    #0,-(sp)        face zero
        move    #0,-(sp)        piste zero
        move    #1,-(sp)        secteur zero
        move    #0,-(sp)        lecteur A
        clr.l   -(sp)
        move.l  #puffer,-(sp)   adresse du tampon
        move    #flopwr,-(sp)   ecrire le boot secteur
        trap    #xbios
        add.l   #20,sp

exit    clr     -(sp)
        trap    #gemdos         retour au bureau

        .bss
puffer  ds.b    512             place pour un secteur
```

```
100   open "R",1,"b:cmdload.prg",16
110   field#1,16 as bin$
120   a$="": for i=1 to 16: read x$: if x$="*"then 150
130   a=val("&H"+x$): s=s+a:a$=a$+chr$(a): next
140   lset bin$=a$: rec=rec+1: put 1,rec: goto 120
150   data 60,1A,00,00,00,84,00,00,00,00,00,00,02,00,00,00
160   data 00,00,00,00,00,00,00,00,00,00,00,00,3F,3C,00,01
170   data 3F,3C,00,00,3F,3C,00,00,3F,3C,00,01,3F,3C,00,00
180   data 42,A7,2F,3C,00,00,00,84,3F,3C,00,08,4E,4E,DF,FC
190   data 00,00,00,14,4A,40,66,54,41,F9,00,00,00,84,11,7C
200   data 00,01,00,1E,3F,3C,00,01,3F,3C,FF,FF,2F,3C,FF,FF
210   data FF,FF,2F,3C,00,00,00,84,3F,3C,00,12,4E,4E,DF,FC
220   data 00,00,00,0E,3F,3C,00,01,3F,3C,00,00,3F,3C,00,00
230   data 3F,3C,00,01,3F,3C,00,00,42,A7,2F,3C,00,00,00,84
240   data 3F,3C,00,09,4E,4E,DF,FC,00,00,00,14,42,67,4E,41
250   data 00,00,00,18,16,1A,28,00,00,00,00,00,00,00,00,00
260   data *
270   close 1:if s<> 8274 then print "Erreurs dans les DATAs!!!": end
280   print "Ok."
```

2.6. C ET LE LANGAGE MACHINE

Dans cette section, nous allons montrer avec un exemple pratique comment relier un sous programme en assembleur avec un programme en C.
Le langage C est certainement beaucoup plus pratique et plus sûr pour écrire un programme important. Mais s'il faut optimiser une partie du programme, il est souvent utile d'écrire des sous programmes en assembleur. Etant donné que le compilateur C crée un programme source en assembleur, on peut penser qu'il est possible de l'optimiser directement. C'est rarement faisable et on n'arrivera presque jamais à résoudre le problème en assembleur. Il faudra plus tard linker le programme assembleur comme n'importe quel programme de la bibliothèque.

Mais comment fixer les paramètres du sous programme à partir du C et récupérer les valeurs ? Il suffit de comprendre le principe. Les paramètres sont d'abord mis sur la pile.

```
int parametre1, parametre2;
long parametre3;
fonction(parametre1,parametre2,parametre3);
```

Le compilateur C génère la séquence suivante à partir de cette recherche de paramètres:

```
move.l parametre3,-(sp)
move.w parametre2,-(sp)
move.w parametre1,-(sp)
jsr    _fonction
addq.l #8,sp
```

Vous voyez donc que la liste des paramètres est traitée à l'envers. Puis on se branche sur la fonction avec un jsr. Le compilateur C écrit un caractère de soulignement avant le nom. Afin que le linker puisse retrouver le nom dans l'assembleur, il doit être inscrit en toutes lettres.

Pour le programme en assembleur, les paramètres sont écrits sur la pile de la manière suivante:

```
8(sp)  long, parametre3
6(sp)  word, parametre2
4(sp)  word, parametre1
0(sp)  long, adresse de retour du branchement jsr
```

Le programmeur doit faire attention à ce que les types des paramètres correspondent lors de l'appel et dans le sous programme. Le compilateur et le linker ne découvriront pas de telles erreurs. Cela est vrai également pour des sous programmes écrits en C.

De plus, il faut faire attention à l'utilisation des registres. Un sous programme en assembleur peut modifier les registres D0-D2 et A0-A2, mais d'autres registres ne doivent pas être modifiés. Si une fonction donne un résultat, il se trouvera dans le registre D0. Dans ce cas, le compilateur part du principe que cette valeur sera du type int ou word, comme dans l'exemple suivant:

```
a=fonction(parametre);
```

Si la fonction donne un résultat sous forme de mot long, il doit être déclaré explicitement avant l'appel de la fonction, p.ex. ainsi:

```
long fonction();
long a;
a=fonction(parametre);
```

Avec ces informations, vous savez tout ce qu'il y a à savoir pour relier des sous programmes en assembleur. Nous prendrons pour exemple l'affichage du catalogue. En plus, le programme vous montre comment les appels de GEMDOS sont traités. A la fin, vous trouverez un petit programme en C qui appelle le sous programme en assembleur. La fonction doit recevoir deux paramètres : le premier précise le lecteur (0=A, 1=B) et le second est une chaîne

vous permettant de choisir par exemple un sous-catalogue. Si vous répondez par une chaîne vide, les autres fichiers du lecteur seront affichés. L'affichage présente 20 fichiers ; le reste peut être obtenu en pressant une touche. Enfin, vous obtenez la place mémoire restante en Koctets.

```
*
*         Affichage du directory
*
*         LE 11/11/85
*

*         Fonctions du BIOS
bios      equ       13          TRAP#
conin     equ       2           entree console
conout    equ       3           sortie console
con       equ       2           No de peripherique

*         Fonctions du GEMDOS
gemdos    equ       1           TRAP#
wrtstr    equ       9           afficher chaine de caracteres
setdrv    equ       $e          selection du lecteur
setdma    equ       $1a         fixer l'adresse dma
getspc    equ       $36         octets libres
sfirst    equ       $4e         recherche du premier
snext     equ       $4f         recherche du suivant

cr        equ       13          retour charriot
lf        equ       10          line feed

filetyp   equ       %11001      type de fichier

wrtchar   move      d0,-(sp)              sortir caractere dans d0
          move      #con,-(sp)
          move      #conout,-(sp)
          trap      #bios
          addq.l    #6,sp
          rts

blank     move.b    #' ',d0               afficher caractere blanc
          bra       wrtchar

newline   lea       crlf(pc),a0           nouvelle ligne
```

```
wrttxt    move.l    a0,-(sp)              adresse du texte
          move      #wrtstr,-(sp)         affichage de la chaine
          trap      #gemdos
          addq.l    #6,sp
          rts

          .globl    _directory            rendre possible l'acces pour C

*         6(sp)     pointeur sur nom de fichier
*         4(sp)     No de lecteur
*         0(sp)     adresse retour

_directory:

          move      4(sp),curdrv          No de lecteur
          move.l    6(sp),a0              nom de fichier
          movem.l   d3-d7/a3-a6,-(sp)             sauvegarde du registre C
          move.l    a0,a3
          move.l    #dmabuf,-(sp)
          move      #setdma,-(sp)         adresse tampon dma
          trap      #gemdos
          addq.l    #6,sp
          move      curdrv,-(sp)
          move      #setdrv,-(sp)         selectionner le lecteur
          trap      #gemdos
          addq.l    #4,sp
          tst.b     (a3)                  nom de fichier en memoire ?
          bne       dir1                  oui
          lea       allfile(pc),a3        nom devient '* *'
dir1      move      #filetyp,-(sp)
          move.l    a3,-(sp)              pointeur sur nom de fichier
          move      #sfirst,-(sp)
          trap      #gemdos
          addq.l    #8,sp
          tst       d0                    fichier en memoire ?
          bne       enddir
dircont   moveq     #20-1,d7              nombre de ligne
```

```
nxtfile   bsr       wrtname              afficher nom de fichier
          move.l    size,d0              taille en octets
          bsr       wrtlng               afficher en nombre decimal
          bsr       blank
          move      date,d3              afficher la
          bsr       wrtdate              date
          bsr       blank                caractere blanc
          move      time,d3              afficher
          bsr       wrttime              temps
          bsr       newline              nouvelle ligne
          move      #snext,-(sp)
          trap      #gemdos              recherche du fichier suivant
          addq.l    #2,sp
          tst       d0                   memorise
          dbne      d7,nxtfile
          bne       enddir               non
          move      #con,-(sp)           attendre qu'une touche soit pressee
          move      #conin,-(sp)
          trap      #bios
          addq.l    #4,sp
          bra       dircont              et continuer

enddir    move      curdrv,-(sp)         drive
          addq      #1,(sp)              1=a, 2=b
          move.l    #buffer,-(sp)
          move      #getspc,-(sp)        place libre sur la disquette
          trap      #gemdos
          addq.l    #8,sp
          move      buffer+2,d0          taille
          bsr       wrt3dec              afficher en decimale avec trois chiffres
          lea       kfree(pc),a0
          bsr       wrttxt
          movem.l   (sp)+,d3-d7/a3-a6          recuperer registre C
return    rts

wrtname   lea       filenam,a6           afficher le nom de fichier formate
          clr       d6
```

```
namloop  move.b   (a6)+,d0        lire caractere
         beq      endnam1         fin du nom ?
         cmp.b    #'.',d0
         beq      extens          continuer si extension
         addq     #1,d6
         bsr      wrtchar         afficher caractere
         bra      namloop
extens   cmp      #9,d6           mettre le nom sur 8 caracteres
         beq      weiter
         addq     #1,d6
         bsr      blank           remplir avec des esoaces
         bra      extens
weiter   move.b   (a6)+,d0        afficher extension
         beq      endnam1
         addq     #1,d6
         bsr      wrtchar
         bra      weiter
endnam1  cmp      #14,d6          fin du nom ?
         beq      return
         bsr      blank           remplir avec des blancs
         addq     #1,d6
         bra      endnam1

wrtdate  bsr      blank           afficher date
         move     d3,d0
         and      #%11111,d0      isoler jour
         bsr      wrt2dec         et l'afficher
         bsr      wrtpkt          caractere de separation '.'
         move     d3,d0
         lsr      #5,d0
         and      #%1111,d0       isoler mois
         bsr      wrt2dec         et l'afficher
         bsr      wrtpkt          caractere de separation '.'
         move     d3,d0
         lsr      #8,d0
         lsr      #1,d0           isoler annee
         add      #80,d0          aditionner debut
```

```
        bra     wrt2dec             et afficher

wrtpkt  move.b  #'.',d0             afficher point
        bra     wrtchar

wrttime bsr     blank               afficher temps
        move    d3,d0
        lsr     #8,d0
        lsr     #3,d0               isoler heures
        bsr     wrt2dec             et afficher
        bsr     wrtcol              caractere de separation '.'
        move    d3,d0
        lsr     #5,d0
        and     #%111111,d0         isoler minutes
        bsr     wrt2dec             et afficher
        bsr     wrtcol              caractere de separation '.'
        move    d3,d0
        and     #%11111,d0
        lsl     #1,d0               isoler secondes
        bra     wrt2dec             et afficher

wrtcol  move.b  #':',d0             afficher deux points
        bra     wrtchar

wrt3dec moveq.l #3,d6               afficher d0 sur 3 chiffres
        clr     d4                  afficher des 0 avant
        ext.l   d0
        bra     wrtlng1

wrt2dec moveq   #2,d6
        ext.l   d0                  afficher d0 sur deux chiffres
        st      d4                  ne pas afficher de 0
        bra     wrtlng1

*               nombre hexa apres decimal dans d0.l
```

```
wrtlng   clr      d4                  flag pour les 0 de remplissage
         moveq    #10,d6

wrtlng1  movem.l  d1-d3/d6-d7,-(sp)
         move.l   d0,d7
wrtdec5  moveq    #1,d2
         move.l   d6,d1
         subq.l   #1,d1
         beq      wrtdec1
wrtdec0  move     d2,d3               10*d3.l d'après d3
         mulu     #10,d3
         swap     d2
         mulu     #10,d2
         swap     d3
         add      d3,d2
         swap     d2
         swap     d3
         move     d3,d2
         subq.l   #1,d1
         bne      wrtdec0
wrtdec1  clr.l    d0
wrtdec3  cmp.l    d2,d7
         blt      wrtdec2
         addq.l   #1,d0
         sub.l    d2,d7
         bra      wrtdec3
wrtdec2  tst.b    d0                  zero ?
         bne      wrtdec4             non, afficher
         tst      d4
         bne      wrtdec4             afficher zero de remplissage
         mp       2#1,d6              dernier chiffre
         beq      wrtdec4             oui, alors afficher zero
         bsr      blank               afficher des blancs en position précedente
         bra      wrtdec6
wrtdec4  add.b    #'0',d0
         bsr      wrtchar             afficher chiffre
         st       d4                  mettre en place le flag
```

```
wrtdec6   subq.l    #1,d6
          bne       wrtdec5
          movem.l   (sp)+,d1-d3/d6-d7
          rts

allfile   dc.b      "*.*",0              tous les fichiers
kfree     dc.b      " K free."
crlf      dc.b      cr,lf,0

          .bss
dmabuf    ds.b      22        tampon dma pour gemdos
time      ds.w      1         heure
date      ds.w      1         date
size      ds.l      1         taille du fichier
filenam   ds.b      14        nom du fichier
curdrv    ds.w      1         No de lecteur courant
buffer    ds.b      16        tampon pour la taille du fichier
```

Le petit programme en C qui suit peut servir de test pour le programme directory.

```
/*
 *   programme test pour l'affichage du catalogue
 *   le 11/11/85
 */
main()
{
  directory (0,"");       /* drive a, tous les fichiers */
  directory (1,"*.prg"); /* drive b, seulement les fichiers prg */
}
```

Si vous appelez le programme source en C 'direc.c' et que vous désignez le sous programme en assembleur par 'dir.s', alors, il faudra utiliser pour la compilation ou l'assemblage lors du linkage la commande suivante :

dir.68k=apstart,direc,dir

```
100  open "R",1,"b:dir.prg",16
110  field#1,16 as bin$
120  a$="": for i=1 to 16: read x$: if x$="*"then 150
130  a=val("&H"+x$): s=s+a:a$=a$+chr$(a): next
140  lset bin$=a$: rec=rec+1: put 1,rec: goto 120
150  data 60,1A,00,00,02,72,00,00,01,64,00,00,04,42,00,00
160  data 00,00,00,00,00,00,00,00,00,00,00,00,2A,4F,2E,7C
170  data 00,00,07,D6,2A,6D,00,04,20,2D,00,0C,D0,AD,00,14
180  data D0,AD,00,1C,D0,BC,00,00,01,00,2F,00,2F,0D,3F,00
190  data 3F,3C,00,4A,4E,41,DF,FC,00,00,00,0C,4E,B9,00,00
200  data 00,4A,2F,3C,00,00,00,00,4E,41,22,2F,00,04,30,3C
210  data 00,C8,4E,42,4E,75,4E,56,FF,FC,2E,BC,00,00,03,CE
220  data 42,67,4E,B9,00,00,00,9A,54,8F,2E,BC,00,00,03,CF
230  data 3F,3C,00,01,4E,B9,00,00,00,9A,54,8F,4E,5E,4E,75
240  data 3F,00,3F,3C,00,02,3F,3C,00,03,4E,4D,5C,8F,4E,75
250  data 10,3C,00,20,60,EA,41,FA,01,E2,2F,08,3F,3C,00,09
260  data 4E,41,5C,8F,4E,75,33,EF,00,04,00,00,08,06,20,6F
270  data 00,06,48,E7,1F,1E,26,48,2F,3C,00,00,07,DA,3F,3C
280  data 00,1A,4E,41,5C,8F,3F,39,00,00,08,06,3F,3C,00,0E
290  data 4E,41,58,8F,4A,13,66,04,47,FA,01,94,3F,3C,00,19
300  data 2F,0B,3F,3C,00,4E,4E,41,50,8F,4A,40,66,46,7E,13
310  data 61,70,20,39,00,00,07,F4,61,00,01,14,61,92,36,39
320  data 00,00,07,F2,61,00,00,9E,61,86,36,39,00,00,07,F0
330  data 61,00,00,C0,61,80,3F,3C,00,4F,4E,41,54,8F,4A,40
340  data 56,CF,FF,CE,66,0E,3F,3C,00,02,3F,3C,00,02,4E,4D
350  data 58,8F,60,BA,3F,39,00,00,08,06,52,57,2F,3C,00,00
360  data 08,08,3F,3C,00,36,4E,41,50,8F,30,39,00,00,08,0A
370  data 61,00,00,AC,41,FA,01,1C,61,00,FF,40,4C,DF,78,F8
380  data 4E,75,4D,F9,00,00,07,F8,42,46,10,1E,67,28,B0,3C
390  data 00,2E,67,08,52,46,61,00,FF,08,60,EE,BC,7C,00,09
400  data 67,08,52,46,61,00,FF,0A,60,F2,10,1E,67,08,52,46
410  data 61,00,FE,EE,60,F4,BC,7C,00,0E,67,C4,61,00,FE,F2
420  data 52,46,60,F2,61,00,FE,EA,30,03,C0,7C,00,1F,61,56
430  data 61,18,30,03,EA,48,C0,7C,00,0F,61,4A,61,0C,30,03
440  data E0,48,E2,48,D0,7C,00,50,60,3C,10,3C,00,2E,60,00
450  data FE,B0,61,00,FE,BC,30,03,E0,48,E6,48,61,28,61,16
460  data 30,03,EA,48,C0,7C,00,3F,61,1C,61,0A,30,03,C0,7C
```

```
470   data 00,1F,E3,48,60,10,10,3C,00,3A,60,00,FE,84,7C,03
480   data 42,44,48,C0,60,0C,7C,02,48,C0,50,C4,60,04,42,44
490   data 7C,0A,48,E7,73,00,2E,00,74,01,22,06,53,81,67,1A
500   data 36,02,C6,FC,00,0A,48,42,C4,FC,00,0A,48,43,D4,43
510   data 48,42,48,43,34,03,53,81,66,E6,42,80,BE,82,6D,06
520   data 52,80,9E,82,60,F6,4A,00,66,10,4A,44,66,0C,BC,7C
530   data 00,01,67,06,61,00,FE,3A,60,0A,D0,3C,00,30,61,00
540   data FE,20,50,C4,53,86,66,B0,4C,DF,00,CE,4E,75,2A,2E
550   data 2A,00,20,4B,20,66,72,65,65,2E,0D,0A,00,00,00,01
560   data 00,02,01,01,02,01,01,00,01,01,02,01,01,01,01,01
570   data 00,00,00,00,00,00,00,00,00,00,01,00,00,01,00,03
580   data 05,00,05,05,00,00,01,01,02,01,00,10,07,01,02,01
590   data 00,00,00,00,00,00,00,00,00,00,01,01,01,02,01,01
600   data 02,01,01,02,01,01,01,01,02,01,01,01,00,00,00,00
610   data 00,00,00,00,00,00,00,00,02,01,01,01,01,01,06,01
620   data 01,04,01,01,01,03,01,02,01,01,04,02,01,08,01,01
630   data 00,00,00,00,00,00,01,01,01,09,01,01,01,01,01,01
640   data 01,00,00,05,01,00,00,00,00,00,00,00,00,00,00,00
650   data 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
660   data 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
670   data 00,00,04,03,00,08,03,00,06,01,00,08,01,00,08,01
680   data 00,04,01,01,03,01,01,00,05,00,01,01,01,00,05,00
690   data 00,01,01,00,01,01,00,00,00,00,00,00,00,00,00,00
700   data 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,02
710   data 02,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
720   data 00,00,00,00,00,00,00,00,00,00,00,00,05,01,00,05
730   data 01,00,01,01,00,01,01,00,02,05,00,06,01,00,02,01
740   data 00,01,01,00,06,05,00,00,00,00,00,01,01,00,01,00
750   data 02,01,00,02,01,01,01,01,01,00,00,00,00,00,00,00
760   data 00,00,00,00,00,00,00,00,00,01,02,03,01,02,01,01
770   data 01,01,01,01,00,01,01,00,01,02,00,2A,2E,50,52,47
780   data 00,00,00,00,00,04,2E,1E,08,08,0A,34,10,0E,2C,0C
790   data *
800   close 1:if s<> 49300 then print "Erreur dans les DATAs!!": end
810   print "Ok."
```

CHAPITRE 3

HARDCOPY EN COULEURS

Un des aspects les plus fascinants de l'ATARI ST sont ses formidables capacités graphiques.
Il n'y a pas besoin de vous dire combien est agréable le mode haute résolution de l'écran monochrome. Vous vous en rendez compte dès la mise en route.

Nous avons cherché une solution permettant de brancher un écran couleur autre que celui livré par ATARI. Nous en avons trouvé une. Vous en saurez plus dans le chapitre suivant.

Nous nous sommes ensuite attachés à l'impression en couleur des images en haute résolution. GEMDOS contient une routine de harcopy. Mais celle ci ne fonctionne qu'avec les imprimantes normales. Les différentes couleurs sont donc imprimées sous forme de nuances de gris. Il existe bien une routine pour les imprimantes couleurs, mais nous n'avons pas encore pu déterminer avec quelles imprimantes elle fonctionne. Nous en avons donc écrit une qui est ainsi construite qu'elle peut être paramètrée en fonction des imprimantes.

N'étant pas encore satisfaits, nous avons cherché à tracer une hardcopy d'écran avec un traceur, à l'aide du programme GEMDRAW. Nous avons rencontré cependant un problème : un des algorithmes à utiliser nous était inconnu. Mais nous avons quand même pu résoudre le problème, comme vous le verrez dans la section 3.3.2.

Afin que vous ne soyez pas limités à saisir deux programmes dont l'utilité peut être restreinte pour vous, nous avons écrit une partie sur la structure de la RAM graphique qui vous sera sûrement très utile pour vos propres réalisations.

De plus, les programmes sont commentés, de telle façon que vous puissiez réaliser simplement vos adaptations.

Afin que vous puissiez utiliser les programmes même si vous ne possédez pas d'assembleur, vous trouverez des chargeurs BASIC.

Nous avons agrémenté le tout de photographies et de copies d'écran afin de montrer le fonctionnement de ces programmes.

ATTENTION : Pour des raisons techniques, ces pages en couleur on dû être réunies. Si vous ne les trouvez pas à coté des applications, voyez les pages en couleur.

Voici maintenant comment connecter un moniteur couleur bon marché à votre ATARI.

3.1. CONNECTION D'UN ECRAN COULEUR

La plupart des écrans couleurs disponibles pour les IBM et compatibles possèdent une entrée RGB, C'est à dire que la couleur est soit allumée soit éteinte, combinée avec un bit de luminosité. Mais l'ATARI dispose de couleurs avec trois niveaux analogiques (trois bits par couleur : mode entrelacé). Il existe bien sûr des moniteurs qui conviennent. Mais leur prix dépasse le budget d'un amateur.

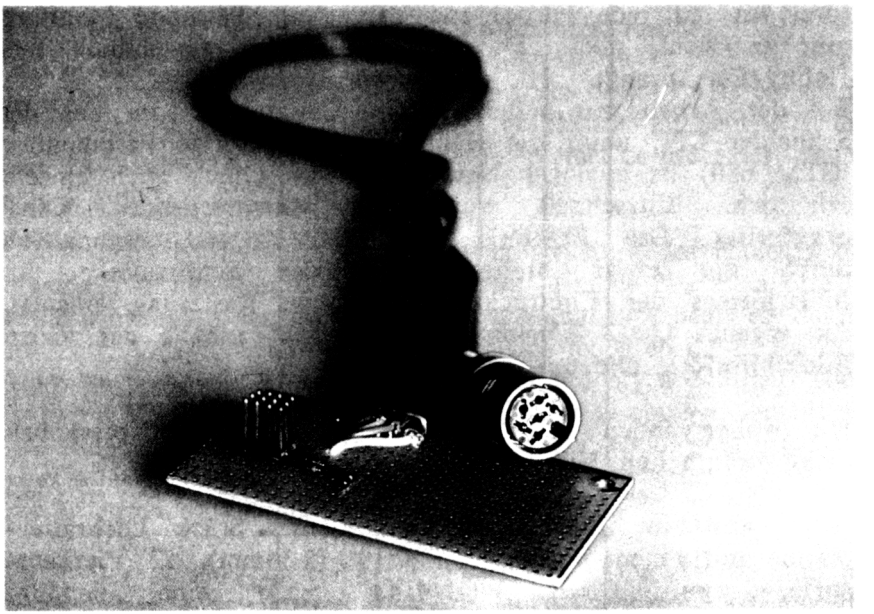
En travaillant avec d'autres ordinateurs personnels, nous avons pu disposer d'un AMSTRAD. Le moniteur couleur du CPC est du même type et par chance disponible seul. Nous avons fait un cable convenant. Le résultat fut merveilleux, bien que les traits soient grossiers étant donnée la largeur du balayage. Vous vous ferez une idée en regardant les photographies.

Nous allons naturellement vous donner le schéma du montage.
Tout ce dont vous avez besoin est d'un morceau d'époxy percé et strié avec des écarts de 2.54 mm (plaque de connection), 12 broches, deux diodes (1n4148), un connecteur DIN 6 broches et un cable à quatres conducteurs.

Le cablage est donné aux figures 3.1-1 et 3.2-2.

Si cela vous intéresse : les diodes servent à mixer les deux signaux de synchronisation de l'ATARI (vertical et horizontal) car le moniteur ne possède qu'une entrée.
Vous pouvez laisser tomber la broche 13 du coté de l'ATARI. Elle ne correspond pas aux trous de la platine.

ATTENTION : Il est possible que l'image tremble de haut en bas. Vous pourrez la rétablir avec le bouton de réglage sur le coté du moniteur. Veuillez à respecter scrupuleusement le shéma de cablage, toute erreur dans les branchements peut endommager le micro-processeur vidéo.



3.1-1

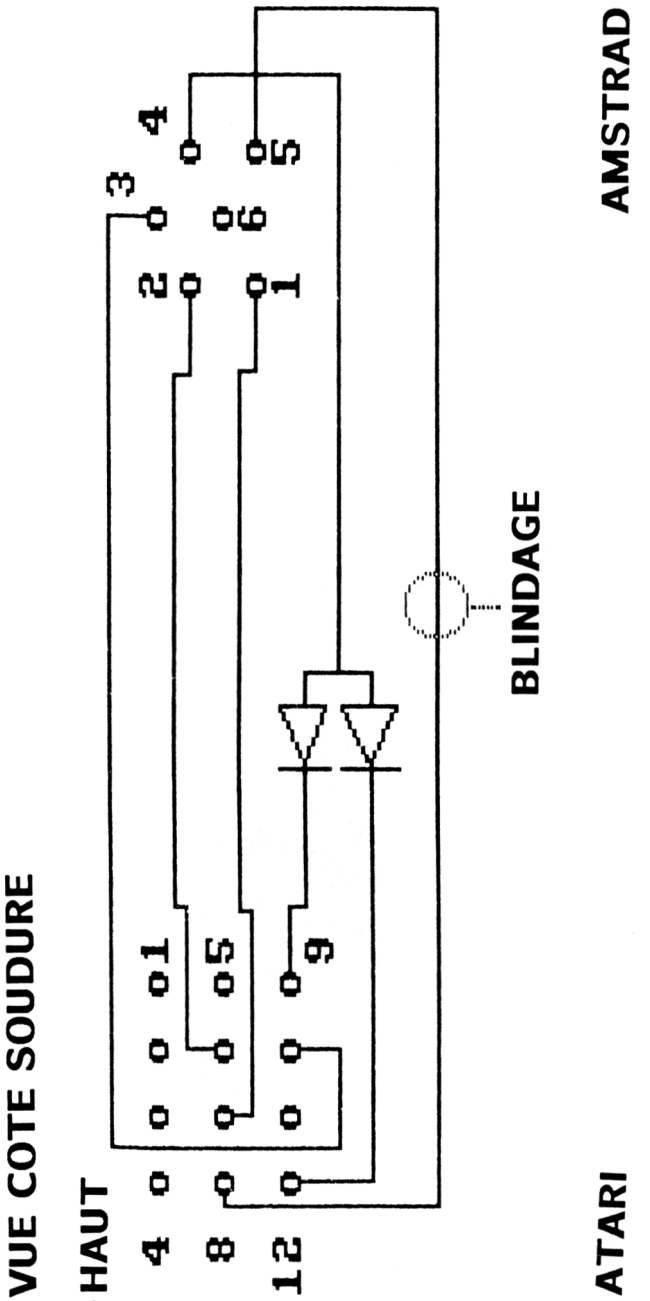

VUE COTE SOUDURE
HAUT
4
8
12
1
5
9
2
3
6
1
4
5
BLINDAGE
AMSTRAD
ATARI

fig. 3.1-2

3.2. STRUCTURE DE LA RAM GRAPHIQUE

La RAM vidéo constitue à la fois la force et la faiblesse de l'ATARI ST. Elle constitue la partie de mémoire utilisée pour la structure des images. Ce secteur de 16K est à destination graphique, cela signifie que le motif de l'écriture normale n'est pas pris dans un générateur de caractères en ROM pour le rafraichissement de l'image, comme c'est le cas pour beaucoup d'ordinateurs.

Les points de l'image restent en RAM vidéo où ils sont copiés à partir d'un générateur de caractères (pour en rester à l'écriture). On peut considérer ce point comme une faiblesse car c'est beaucoup plus lent qu'un générateur de caractères en ROM, ce qui est sensible en particulier lors du scrolling de l'écran. Il y a en effet une différence entre le déplacement de 2K et de 16K de mémoire. On s'en rendrait compte si le 68000 n'était pas aussi rapide.

Mais cet aspect constitue également une force de l'ATARI car étant donné que la RAM est déjà orientée graphisme, on peut afficher simplement des points où on veut ce qui permet de créer des images simplement.

Nous allons vous montrer comment la structure des couleurs est organisée. Le plus simple est évidemment le mode 2, c'est à dire le mode haute résolution qui est en noir et blanc. Le schéma est donné en 3.2-1. Vous avez une hardcopy dans ce mode en 3.2-2. La ram vidéo est organisée en mots. Etant donné qu'il n'y a que deux couleurs, un bit suffit pour donner le numéro de la couleur. Un bit en ram vidéo correspond à un point à l'écran. La mémoire est organisée ainsi : le bit de poids fort du premier mot représente le point de l'écran en haut à gauche.

Il est plus difficile de comprendre le mode 1. On dispose dans ce mode de 640*200 points en quatre couleurs. La question est alors de savoir sous quel mode les couleurs sont mémorisées. En effet, il n'est plus possible d'appliquer le principe 1 bit = une couleur.

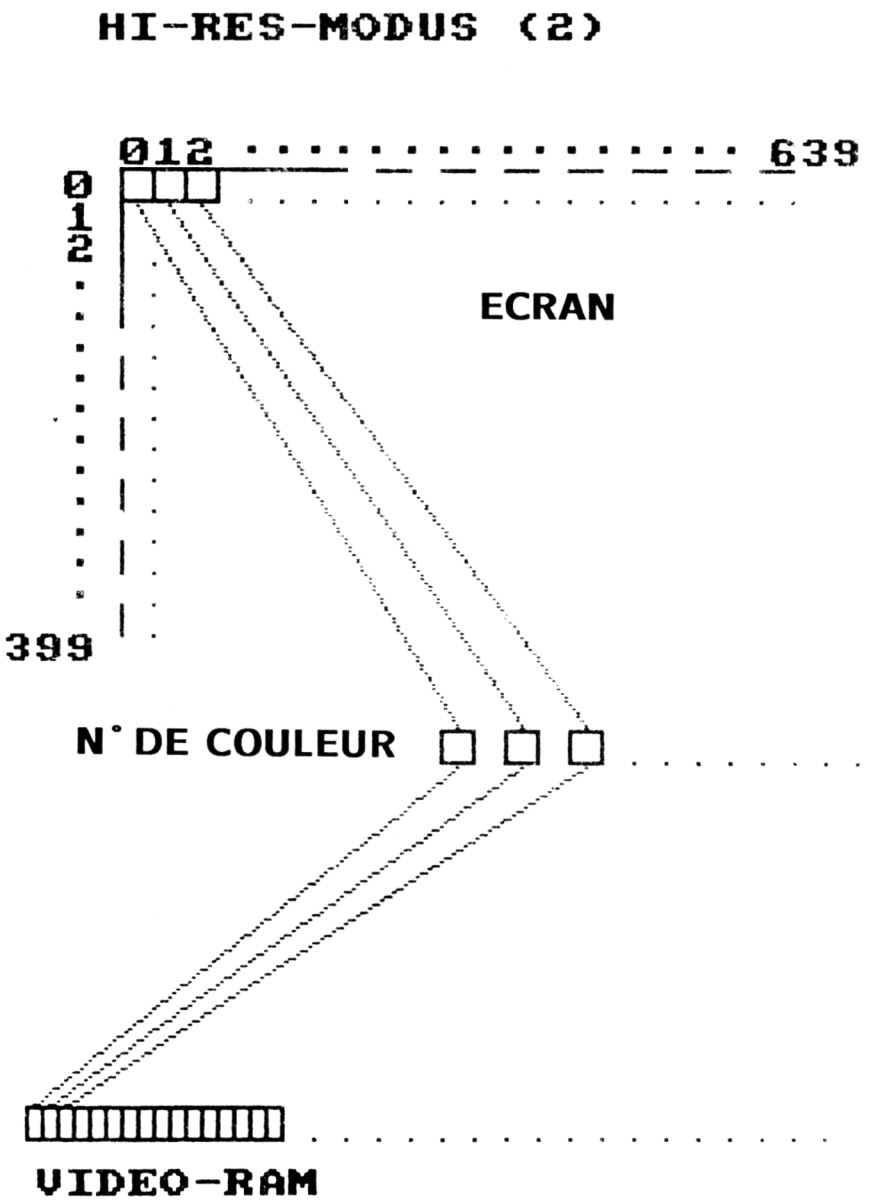


fig. 3.2-1

Bureau Fichier Visualisation Options
A:
DISKSTATIO
B:
DISKSTATIO
B:\
264903 octets utilisés par 16 objets
FAFIERKORE
ED.TTP
MAI.BAK
MAI.S
MAI.TOS
MOND.S
MULTI.BAK
MULTI.DMP
MULTI.LST
MULTI.S
MULTI.TOS
FLOT.BAK
FLOT.BAS
FLOT.DMP
FLOT.LST
FLOT.S
FLOT.TOS

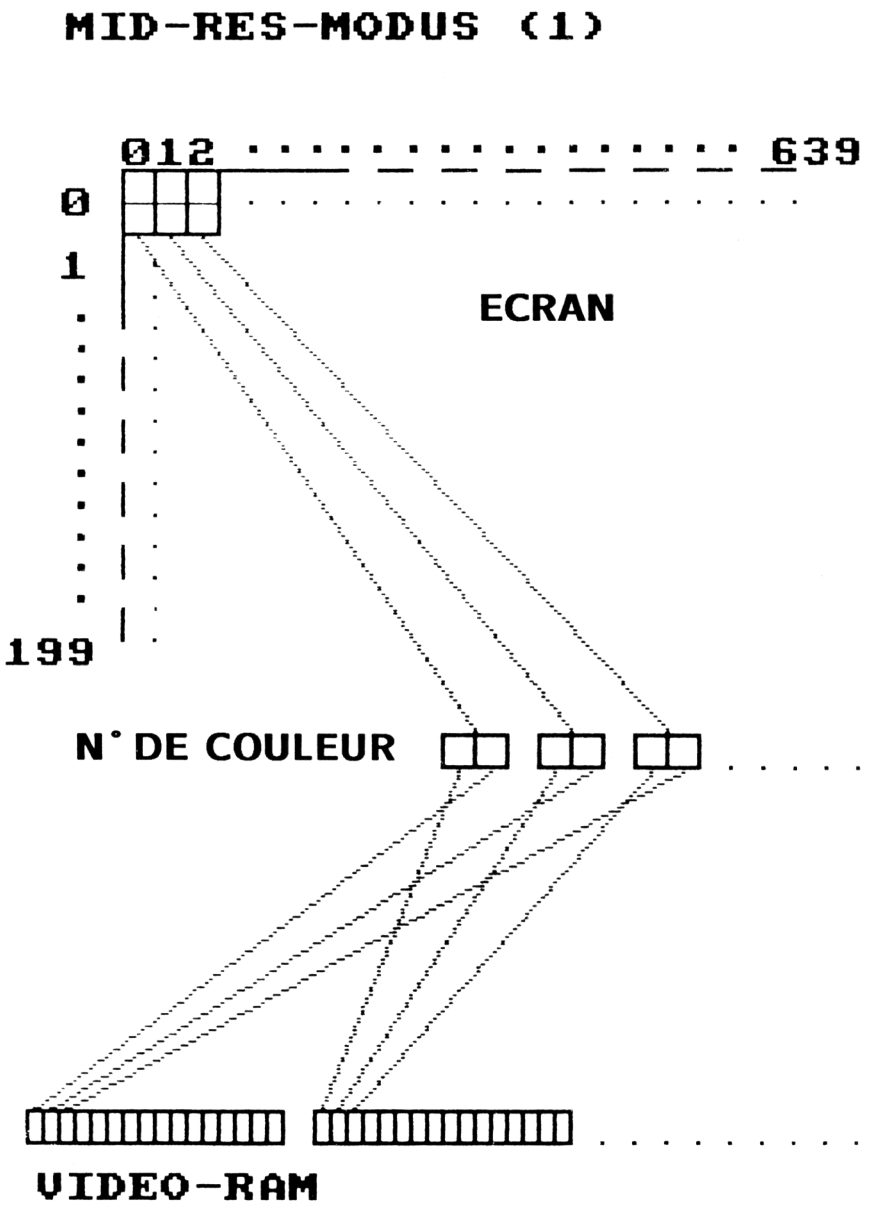


fig. 3.2-3

En fait, deux bits définissent la couleur d'un point, car ils indiquent les valeurs 0 à 3. Comme vous pouvez le voir en 3.2-3, ces bits ne sont pas dans le même mot mais deux bits de deux mots voisins forment une couleur. Afin que l'écran soit rempli en hauteur, les points sont un peu alongés, si bien qu'un point est en fait un trait vertical.

La figure 3.2-4 dans les pages en couleur montre que les lettres sont un peu étirées en hauteur.

Le mode 0, enfin, qui possède la résolution la plus basse avec 320*200 points, mais qui permet d'afficher 16 couleurs, est construit ainsi : quatre bits définissent un point et donnent une valeur de coloris allant de 0 à 15. Les bits sont inscrits dans ce cas dans quatre mots successifs. Faites attention au fait que le bit du premier mot est le bit de poids faible de la couleur (voir fig. 3.2-5).

Pour construire l'image, les points ne sont pas alongés seulement dans la longueur mais aussi en horizontal. Comme vous le voyez en fig. 3.2-6, les proportions sont de nouveau respectées.
Nous en avons fini avec la ram vidéo.

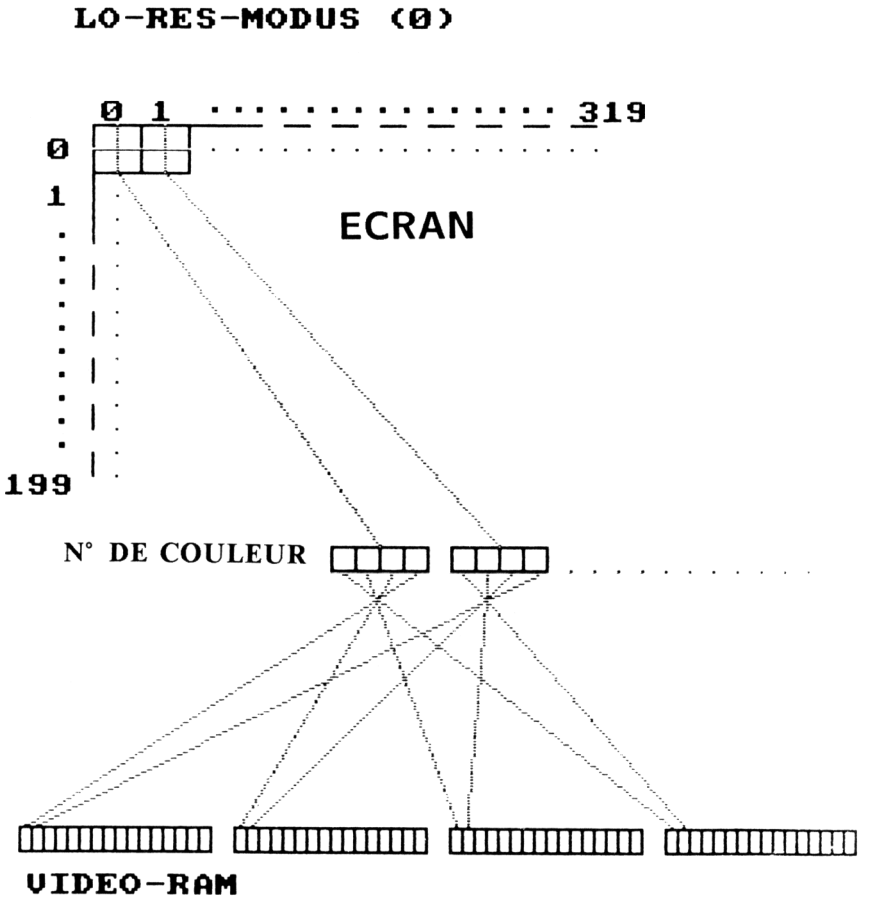


fig. 3.2-5

3.3. HARDCOPY

Comme nous l'avions indiqué dans l'introduction de ce chapitre, nous allons vous donner deux moyen pour sortir une hardcopy : une méthode sur imprimante matricielle et une sur traceur.

Les programmes sont écrits pour des périphériques courants, si bien que vous ne devriez avoir aucun problème pour les utiliser. Nous avons choisi des appareils EPSON parce qu'ils sont très courants et parce qu'il ne videront pas trop votre bourse.

Naturellement vous pouvez utiliser d'autres périphériques. Les programmes sont construits de telle façon que vous ne devrez adapter que quelques constantes pour les faire correspondre.
Les programmes sont mis en route très simplement. C'est vrai aussi bien pour les versions BASIC que pour les versions en assembleur. A la mise en route, rien ne se passe car le programme a été simplement copié derrière la RAM vidéo (il existe 768 octets de libres en cet endroit) et il y restera tant que l'ordinateur sera allumé. La HARCOPY est activée si vous pressez les touches ALTERNATE et HELP. Le programme remplace en pratique l'ancienne routine.

Nous vous proposons deux procédés car les imprimantes et les traceurs ont leur propres forces dans certains domaines. Ainsi, une imprimante matricielle peut rendre des nuances de couleurs assez fines en dosant les espacements mais elle ne permet pas de tracer un trait continu. Ceci est possible avec un traceur.

Les application sont donc différentes : les imprimantes permettent d'imprimer des dessins avec de nombreuses nuances de couleurs alors que les traceurs sont très pratiques pour la reproduction de dessins techniques dans peu de couleurs (voire une seule).

Les résultats que vous trouverez dans les pages en couleur vous permettront de vous décider. Pour chaque résolution, vous avez au moins une photographie et une recopie d'écran.

Les deux programmes se différencient par leur adressage de la mémoire vidéo.
Elle est adressée physiquement en ce qui concerne l'imprimante, mais pour ce qui est de la version traceur, on y a accès par l'émulateur line A. Vous apprendrez ainsi à manipuler ces deux possibilités.

Encore un mot sur les images : elles proviennent presque toutes de programmes de démonstration d'ATARI. Nous avons pu les publier grâce à l'autorisation amicale d'ATARI que nous remercions.

3.3.1. IMPRIMANTE MATRICIELLE

Nous n'avions pas pensé qu'une simple recopie d'écran pouvait poser des problèmes aussi profonds. L'ordonnancement des couleurs n'en est pas le plus petit.
Nous nous sommes longtemps posé la question de savoir si une couleur 'élevée' devait être imprimée en foncé ou en clair. Le problème dont cela découle est le suivant : l'écran vide de couleurs apparaît foncé alors que le papier est clair.
Si une couleur est plus forte à l'écran (et par conséquent plus claire), doit-elle être également plus forte (donc plus foncée) sur le papier ? Ceci ne rendrait pas les effets d'ombre (cf. fig. 3.3.1-7).

Nous nous sommes donc décidés pour imprimer les couleurs claires en clair sur le papier. Bien entendu, cela peut aussi donner des images de mauvaise qualité.
Vous en avez un exemple en fig. 3.3.1-3. Si vous êtes de l'avis contraire, il vous suffit de faire une rotation du masque dans l'autre sens dans le programme. En fig. 3.2-4 nous nous sommes trompé. Les couleurs étaient différentes sur la photographie.

Naturellement, aucun problème ne se pose en mode 2 car il n'y a que du noir et blanc. Un exemple excellent est donné en fig. 3.3.1-1.
Venons en au programme. Il tourne tel quel sur une imprimante EPSON JX-80.

C'est la version couleur de la FX-80. Elle possède un large ruban en couleur sur lequel se trouvent les trois couleurs de base et le noir situées par bandes les unes en dessous des autres. En mode direct, on dispose de 7 couleurs car l'imprimante mixe les couleurs automatiquement. Nous nous sommes limités à ces coloris.

Pour changer de teinte, un moteur déplace la couleur désirée au niveau de la tête d'écriture. Le programme calcule ce processus. On recherche une couleur donnée sur chaque ligne de l'écran et on imprime la ligne complète dans cette couleur. Cela prendrait trop de temps de changer de couleur de temps en temps sur la ligne.
Si la couleur recherchée n'est pas sur la ligne, on ne change pas de couleur.
Cela permet de gagner du temps si il y a peu de couleurs.

Vous devrez faire preuve de compréhension si vous voulez utiliser les 16 couleurs en mode 0. Cela peut durer une demi-heure, comme par exemple la figure 3.3.1-5

A la page suivante, vous trouverez un listing assembleur du programme HARDCOPY.
Il est suivi de quelques remarques.

fig. 3.3.1-1

```
*          org     $cba

gemdos     equ     1
xbios      equ     14
prchar     equ     5
sbase      equ     2
getres     equ     4
aff        equ     -2        nombre de couleurs
afc        equ     -4        compteur de couleurs
pwf        equ     -6        mots/pixel
hmf        equ     -8        multiplicateur horizontal
vmf        equ     -10       miltiplicateur vertical
zbl        equ     -14       ligne de base
zwf        equ     -16       nbre de mots/ligne
zwc        equ     -18       compteur nbre de mots
znf        equ     -20       nbre aiguilles/ligne
znc        equ     -22       compteur nbre de lignes
baf        equ     -24       position verticale
zzc        equ     -26       compteur lignes
zol        equ     -30       début de lignes
ab         equ     -31       bit trouvé
fl         equ     -32       flag de div.
*          bit     0         bit correspondant à la couleur trouvé
*          bit     1         0=test / 1=imprime
ctf        equ     -48       table de couleurs
maf        equ     -64       n0 de masque
pflag      equ     $4ee      flag alt/help
super      equ     32        mode superviseur
stcol      equ     7         setcolor

dummy      lea     dummy,a0        dummy pour chargements sans intéret
           clr.l   -(a7)
           move.w  #super,-(a7)
           trap    #gemdos
           addq.l  #6,a7
           move.l  d0,d6
           move.w  #sbase,-(a7)
           trap    #xbios
```

```
            addq.l  #2,a7

            movea.l d0,a0
            adda.w  #$7d00,a0
            lea     (a0),a2
            lea     start(pc),a1
            move.l  #fin-start-1,d0
reloc        move.b  (a1)+,(a0)+
            dbra    d0,reloc

            movea.l $456,a0
            adda    #28,a0
            move.l  a2,(a0)
            move.l  d6,-(a7)
            move.w  #super,-(a7)
            trap    #gemdos
            addq.l  #6,a7
*           rts                     si appel du basic
            clr.l   -(a7)
            trap    #gemdos
start:
            tst     pflag           hardcopy désirée   ?
            beq     st0             oui>
            rts


*************************************************************************
*                                                                       *
*       initialisation des paramètres                                   *
*                                                                       *
*************************************************************************

st0         link    a6,#-66         faire place pour secteur de travail
            move.w  #sbase,-(a7)    adresse base de l'écran
            trap    #xbios          mémorisée
            addq.l  #2,a7
            move.l  d0,zbl(a6)
            move.w  #getres,-(a7)
```

```
        trap    #xbios

        addq.l  #2,a7
        lsl.w   #1,d0
        move.w  d0,(a6)
        lea     aft(pc),a1
        move.w  0(a1,d0.w),aff(a6)
        moveq   #1,d7           préparation de la couleur et du masque
        moveq   #8,d0           si en hi-res
        move.b  #7,ctf(a6)
        clr.b   ctf+1(a6)
        cmpi.w  #1,aff(a6)      hi-res ?
        beq     st52            oui>
        move.w  aff(a6),d7
st1     move.w  #-1,-(a7)
        move.w  d7,-(a7)
        move.w  #stcol,-(a7)
        trap    #xbios          couleur dans d0
        addq.l  #6,a7
        clr.w   d4
        clr.b   maf(a6,d7.w)
        move.w  d0,d1
        moveq   #2,d5
        lsl.w   #4,d1
st10    lsr.b   #4,d1
        or.b    d1,d4           clair > d4
        lsr.w   #4,d1
        dbra    d5,st10

        move.b  d4,maf(a6,d7.w)
        cmpi.b  #1,d4           seuil-noir dépassé?
        bls     st22            oui>
        moveq   #2,d6
        move    #$444,d5        charger masque
st11    move    d5,d3
        and     d0,d3           chercher bit de poids fort
        bne     st12            trouvé >
        lsr     #1,d5           effacer masque
```

```
            dbra    d6,st11


st12        moveq   #2,d4
            clr.w   d5
st2         andi    #$7ff,d3        fixer couleurs (dans d5)
            cmpi.w  #$ff,d3
            bls     st21
            bset.l  d4,d5
st21        lsl.w   #4,d3
            dbra    d4,st2

            cmpi.b  #7,d5           blanc ?
            bne     st5             non >
            cmpi.b  #5,maf(a6,d7)   blanc clair ?
            bhi     st5             oui>
            addq.b  #2,maf(a6,d7)   mettre masque plus fin
st22        clr     d5              et en noir
st5         move.b  d5,ctf(a6,d7.w)
            cmpi.b  #6,d5           jaune?
            bne     st50            non  >
            subq.b  #2,maf(a6,d7)   masque plus gras
st50        dbra    d7,st1

            moveq   #15,d7
st51        moveq   #8,d0
            cmpi.b  #3,maf(a6,d7)   clarté> seuil inférieur?
            bls     st52            non  >
            lsr     #1,d0
            cmpi.b  #6,maf(a6,d7.w) clarté > seuil supérieur?
            bls     st52            non >
            clr     d0
st52        move.b  d0,maf(a6,d7.w)
            dbra    d7,st51

            move.w  (a6),d0
            lea     pwt(pc),a1
            move.w  0(a1,d0.w),pwf(a6)
```

```
        lea     hmt(pc),a1

        move.w  0(a1,d0.w),hmf(a6)
        lea     vmt(pc),a1
        move.w  0(a1,d0.w),vmf(a6)
        lea     zwt(pc),a1
        move.w  0(a1,d0.w),zwf(a6)
        lea     znt(pc),a1
        move.w  0(a1,d0.w),znf(a6)
        lea     bat(pc),a1
        move.w  0(a1,d0.w),baf(a6)
        move.w  #50,zzc(a6)
        clr.b   fl(a6)
        bra     nl0


*************************************************************************
*                                                                       *
*       ligne suivante                                                  *
*                                                                       *
*************************************************************************

nl:
        subq.w  #1,zzc(a6)      compteur de ligne arrivé à zéro?
        beq     exit            oui>
        move.l  zbl(a6),d7      augmenter la
        addi.l  #640,d7         ligne de base
        move.l  d7,zbl(a6)      d'une ligne
nl0     lea     lftab(pc),a5    retour chariot
        moveq   #4,d7           sur
        bsr     lf              l'imprimante
        move.w  aff(a6),afc(a6) compteur de couleurs
        movea.l zbl(a6),a3      base de la ligne
        bra     sl0
```

```
*************************************************************************

*                                                                       *
*       couleur suivante                                                *
*                                                                       *
*************************************************************************

sl:
          tst.w   pflag           arrèter harcopy?
          bne     exit            oui>
          subq.w  #1,afc(a6)      compteur de couleurs à zéro?
          bmi     nl              oui> nouvelle ligne
          bra     sl0
sl00      bchg.b  #1,fl(a6)       dernière boucle seulement un test?
          bne     sl              non  > on a imprimé
          btst.b  #0,fl(a6)       point trouvé dans la ligne ?
          beq     sl00            non >
          lea     ctf(a6),a1
          adda.w  afc(a6),a1
          clr.w   d6
          move.b  (a1),d6
          cmpi.b  #7,d6           blanc ?
          beq     sl00            oui> pas d'impression
          lea     pre1(pc),a5     échange de couleurs
          moveq   #3,d7           sur
          bsr     lf              imprimante
          lea     ct(pc),a1
          move.b  0(a1,d6.w),d0
          bsr     chout
          lea     pre2(pc),a5
          moveq   #5,d7
          bsr     lf
sl0       move.w  zwf(a6),zwc(a6) nombre de mots/ligne
          bclr.b  #0,fl(a6)
          lea     0,a4
          move.w  afc(a6),d7      numéro de couleur cherché
          clr.w   d0
          move.b  maf(a6,d7.w),d0 charger masque
```

```
          lea     mask(pc),a0

          move.l  0(a0,d0.w),d2
          bra     sw0


*************************************************************************
*                                                                       *
*       mot suivant                                                     *
*                                                                       *
*************************************************************************

sw:
          subq.w  #1,zwc(a6)      compteur de mots à zéro?
          beq     sl00            oui>
          movea.l zol(a6),a4      début de ligne
          adda.w  pwf(a6),a4      mots/pixel
          adda.w  pwf(a6),a4      *2
sw0       move.w  #$8000,d5       masque de bits pour test
          move.l  a4,zol(a6)      sauvegarder début de ligne
          bra     sb0


*************************************************************************
*                                                                       *
*       bit suivant                                                     *
*                                                                       *
*************************************************************************

sb:
          lsr.w   #1,d5           tous les bits du mots sont traités?
          beq     sw              oui>
sb0       move.w  znf(a6),znc(a6) anzahl nadeln/zeile
          clr.b   d4
          movea.l zol(a6),a4
          bra     tb
```

```
*************************************************************************

*                                                                       *
*       aiguille suivante                                               *
*                                                                       *
*************************************************************************

bs:
          clr.l   d7
          move.w  vmf(a6),d7      multiplicateur vertical
          subq    #1,d7
bs0       lsl.b   #1,d4
          or.b    ab(a6),d4
          dbra    d7,bs0
          adda.w  baf(a6),a4      position verticale du point
          subq.w  #1,znc(a6)      compteur d'aiguilles à zéro?
          bne     tb              non  > tester point
          tst.b   d4              y-avait-il un point?
          beq     bs00            non  >
          bset.b  #0,fl(a6)
bs00      btst.b  #1,fl(a6)       faut-il imprimer?
          beq     sb              non  >
          clr.l   d7
          move.w  hmf(a6),d7      multiplicateur horizontal
          subq    #1,d7
bs1       move.b  d4,d0
          and.b   d2,d0           masquer octet
          bsr     chout           et imprimer
          ror.l   #8,d2           rotation masque de raster
          dbra    d7,bs1
          bra     sb


*************************************************************************
*                                                                       *
*       test de bit                                                     *
*                                                                       *
*************************************************************************
```

```
        lea     0(a3,a4),a0
        lea     0(a0,d0.w),a5
tb1     lsl.b   #1,d3           prendre bits pour numéro de couleur
        subq.l  #2,a5
        move.w  (a5),d7
        and.w   d5,d7           bit positionné?
        beq     tb2             non  >
        bset.l  #0,d3
tb2     dbra    d6,tb1
        clr.b   ab(a6)
        cmp.w   afc(a6),d3      numéro de couleur recherché?
        bne     tb3             non  >
        bset.b  #0,ab(a6)       marquer le points car trouvé
tb3     bra     bs

*****************************************************************
*                                                               *
*       sortie                                                  *
*                                                               *
*****************************************************************

exit:
        unlk    a6              secteur de travail libre
        move.w  #-1,pflag       hardcopy finie
        rts

*****************************************************************
*                                                               *
*       imprimr chaine en (a5) avec compteur en d7              *
*                                                               *
*****************************************************************
```

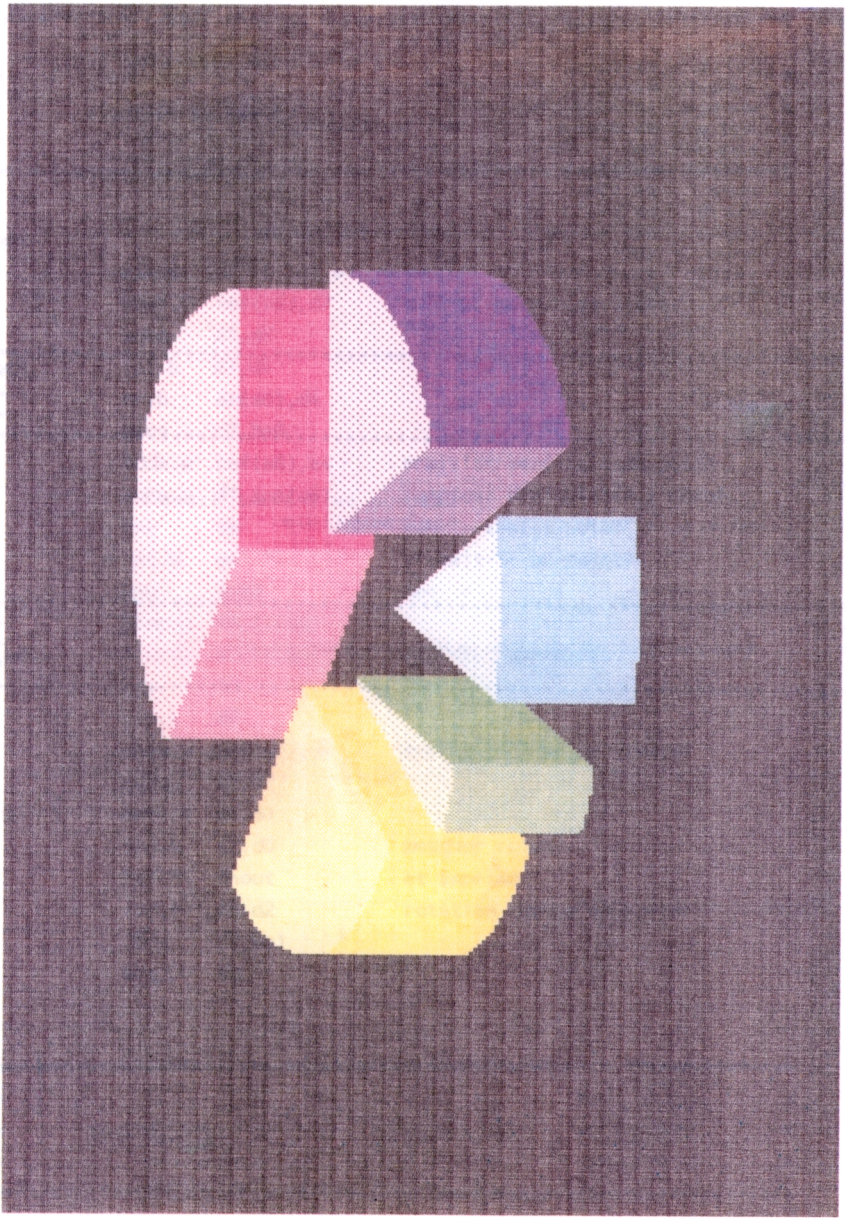


fig. 3.1-2

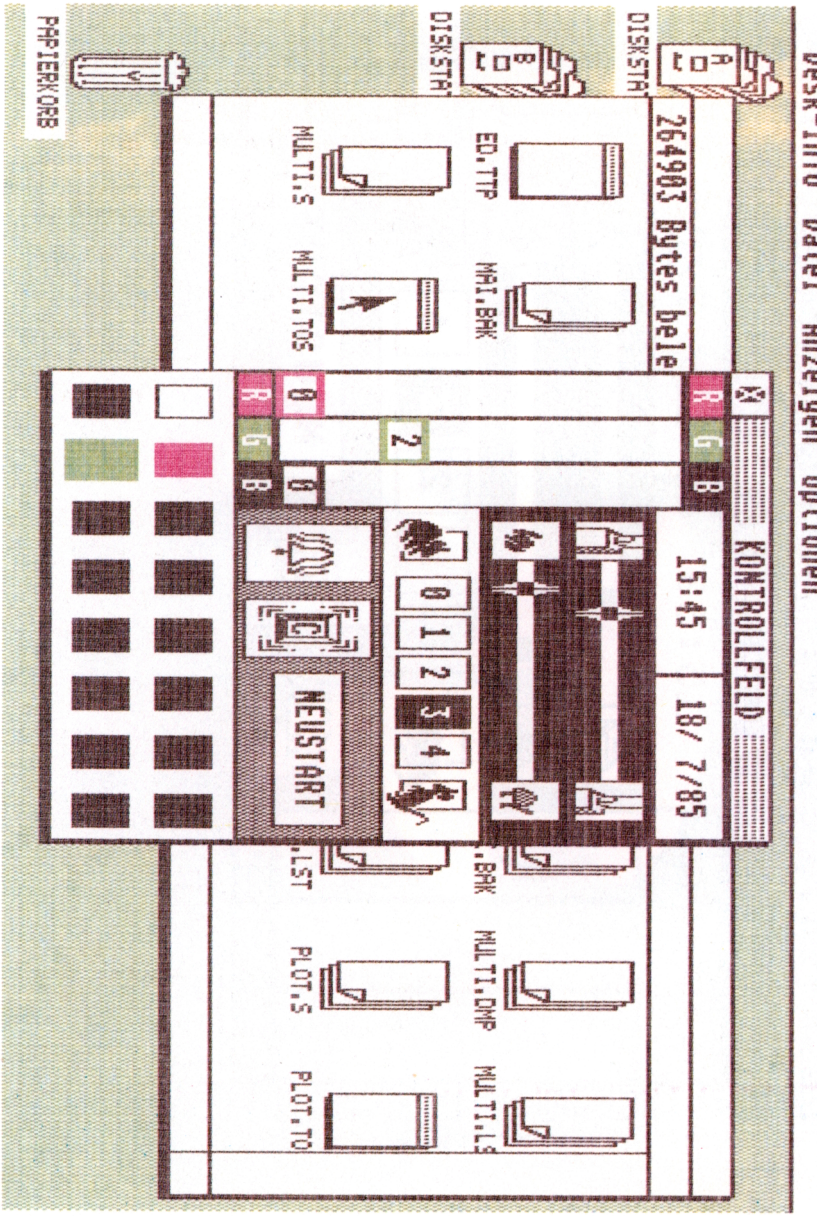

Desk-Info Datei Anzeigen Optionen
KONTROLLFELD
15:45
18/ 7/85
264903 Bytes bele
DISKSTA
DISKSTA
PAPIERKORB
ED.TTP
MAI.BAK
MULTI.S
MULTI.TOS
MULTI.LST
PLOT.S
PLOT.TO
MULTI.DMP
MULTI.LS
NEUSTART

fig. 3.2-4

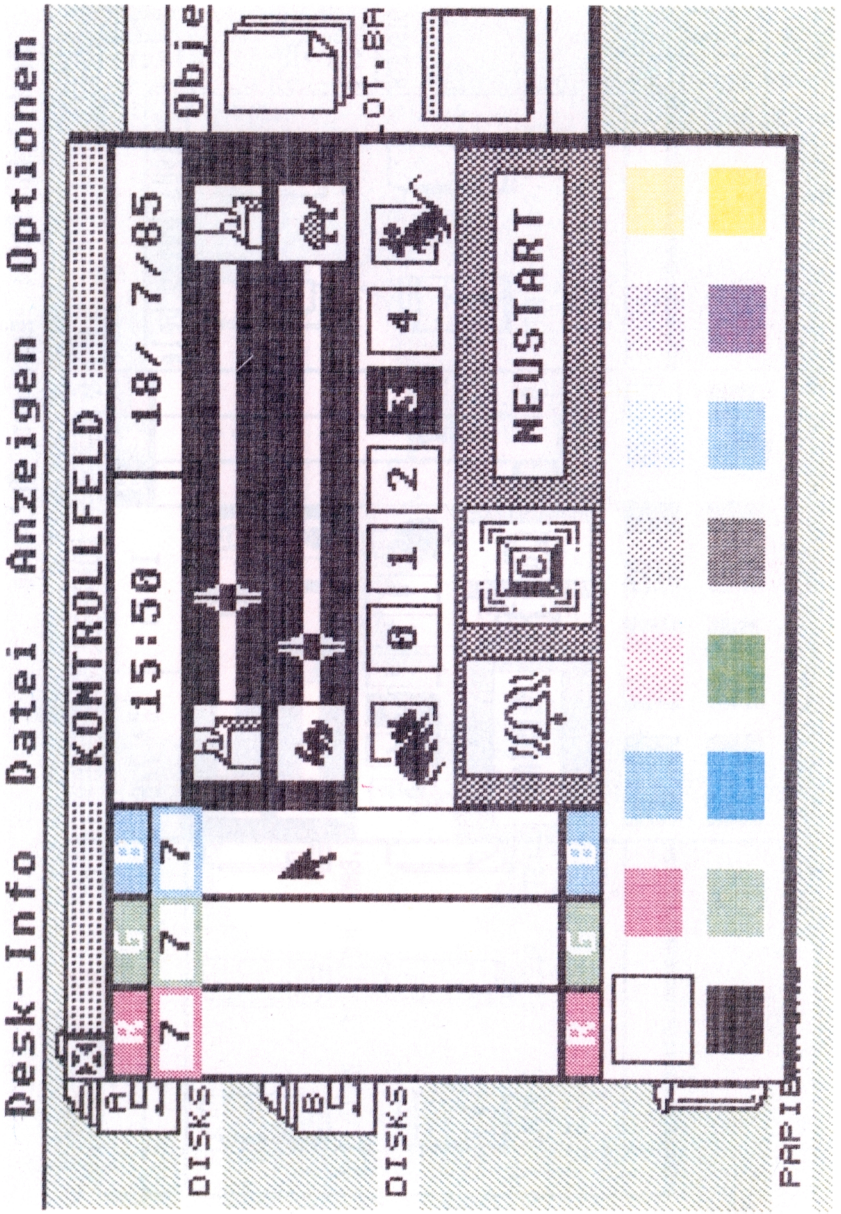

Desk-Info Datei Anzeigen Optionen
KONTROLLFELD
15:50
18/ 7/85
Obje
OT.BA
NEUSTART
0 1 2 3 4
A
B
C
DISKS
DISKS
PAPIE

fig. 3.2-6

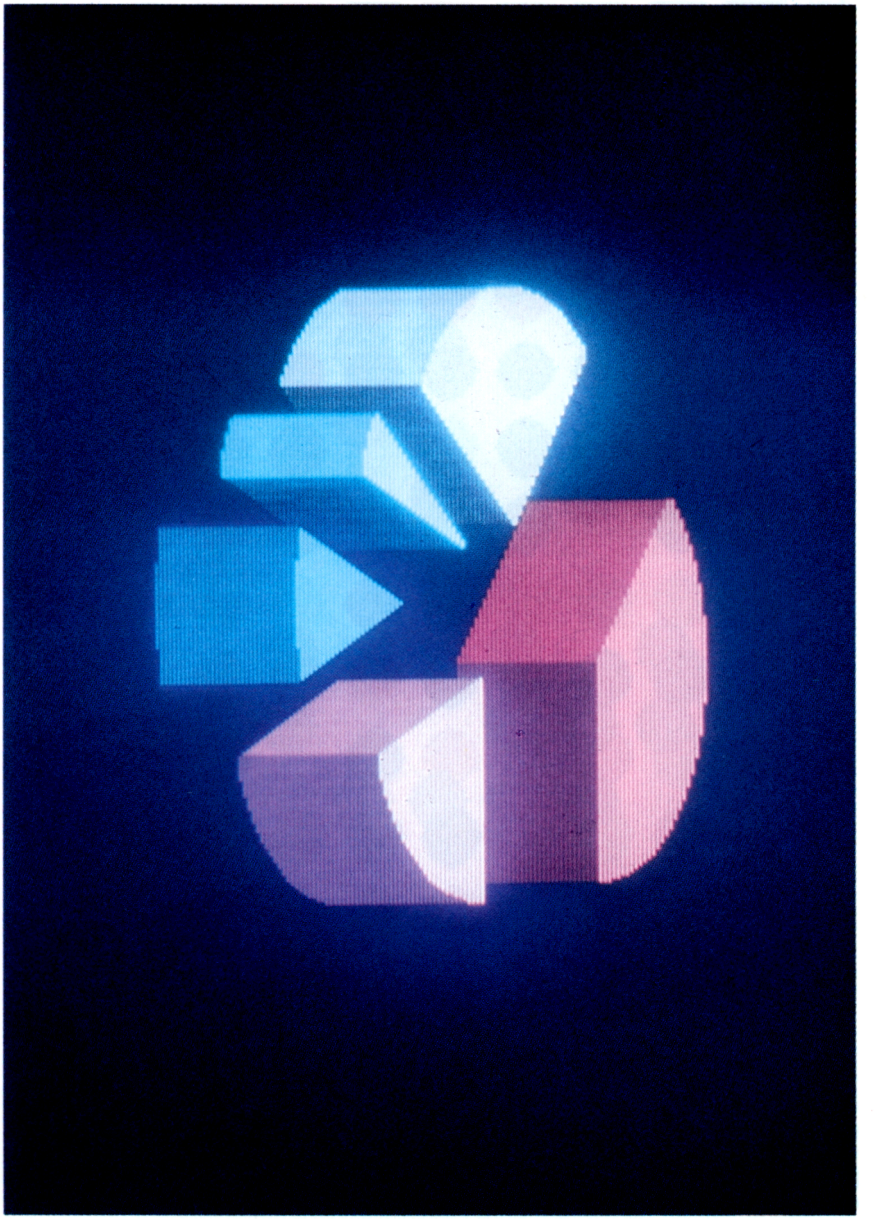


fig. 3.1-1

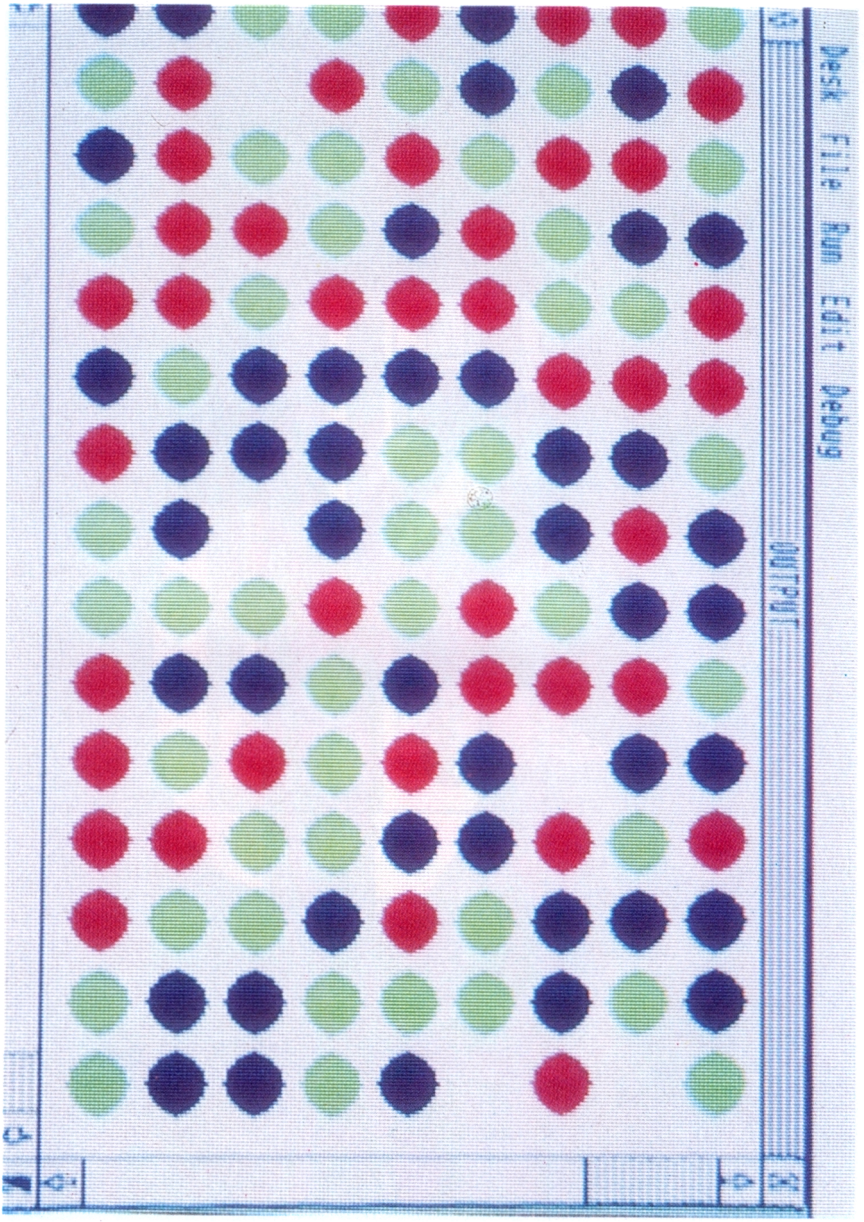


fig. 3.3.1-2

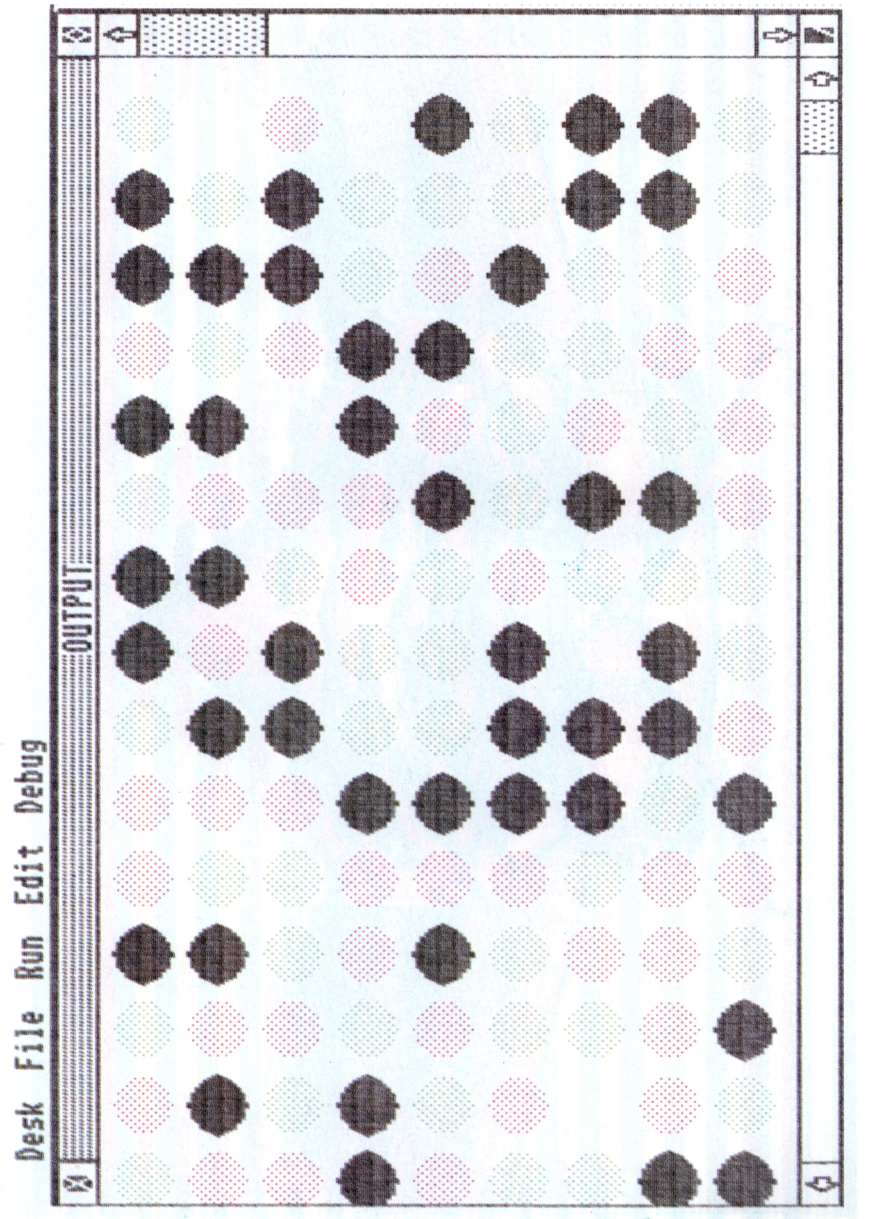


fig. 3.3.1-3

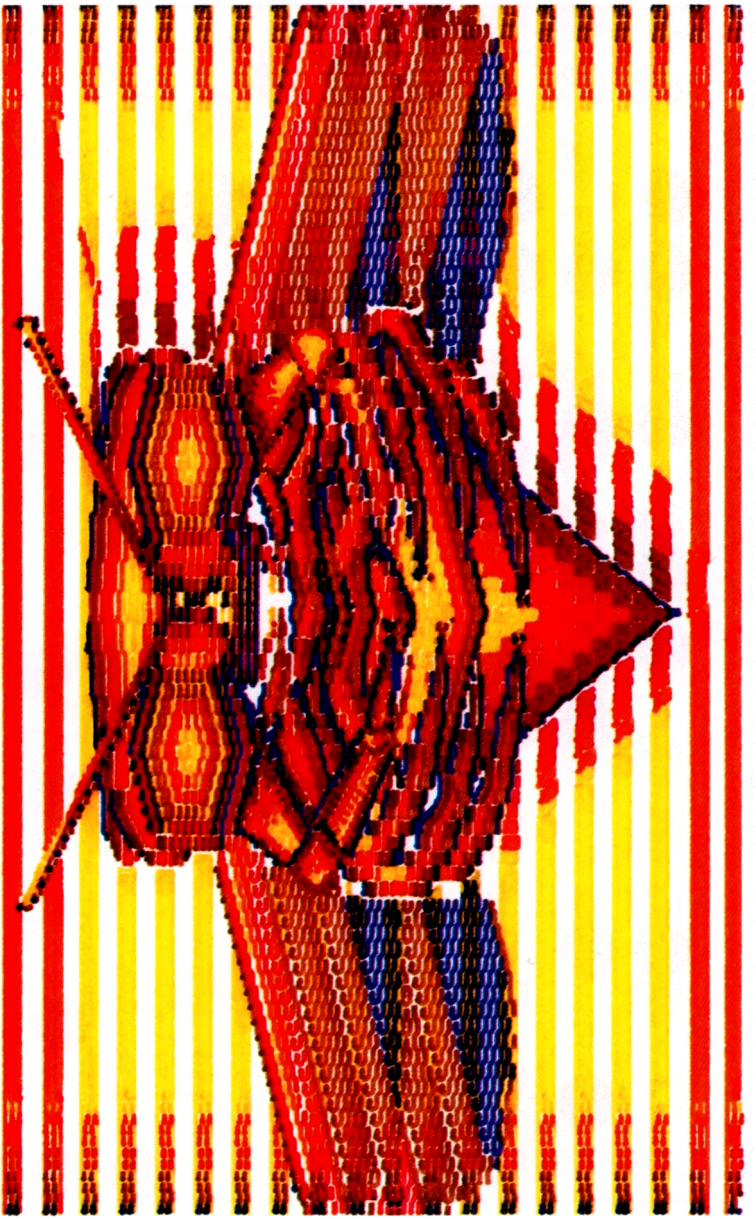


fig. 3.3.1-4

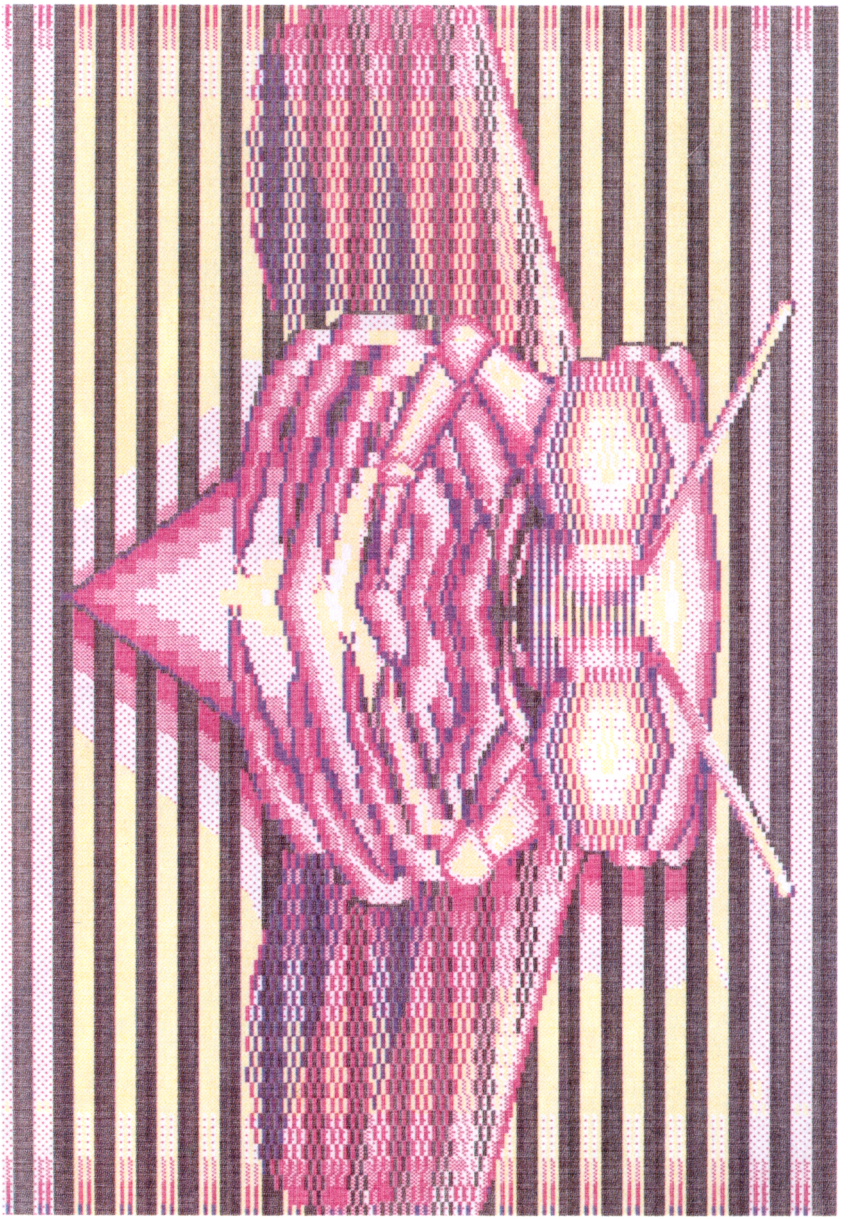


fig. 3.3.1-5

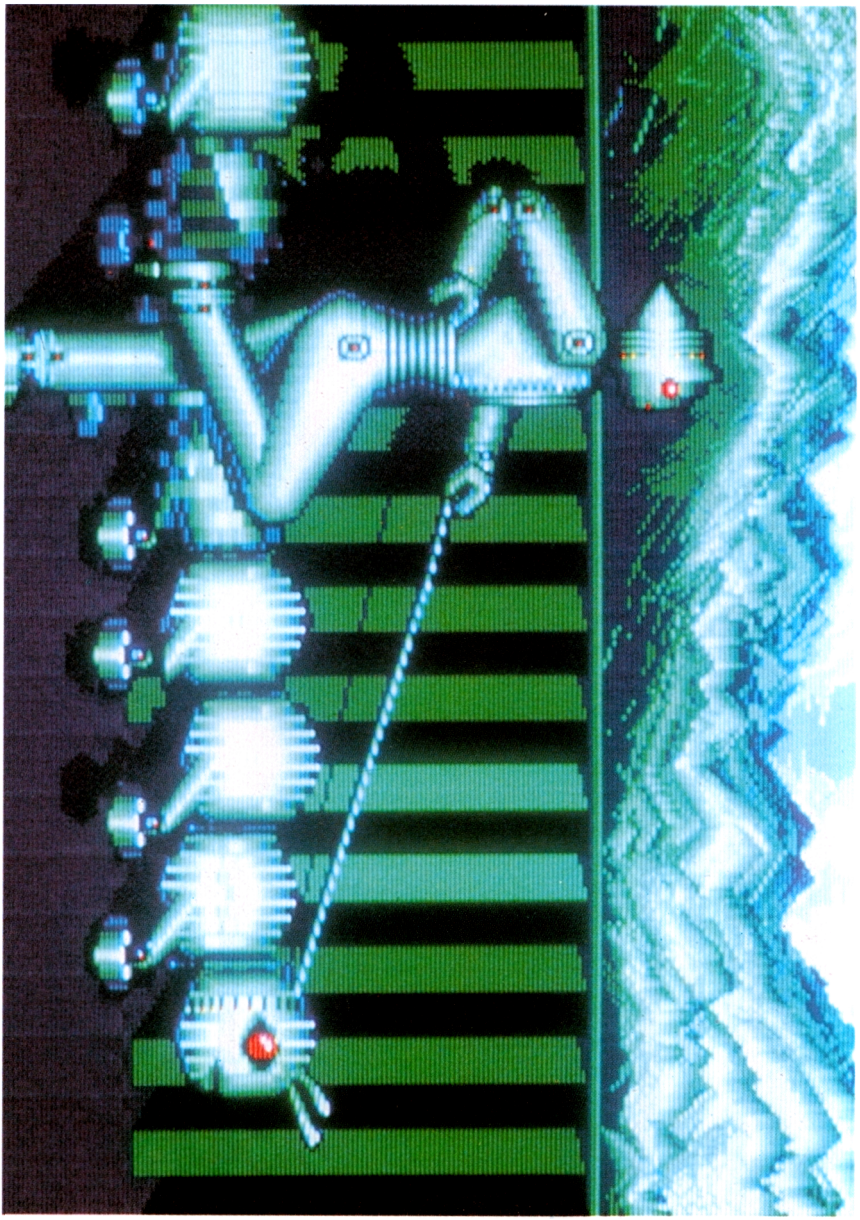


fig. 3.3.1-6

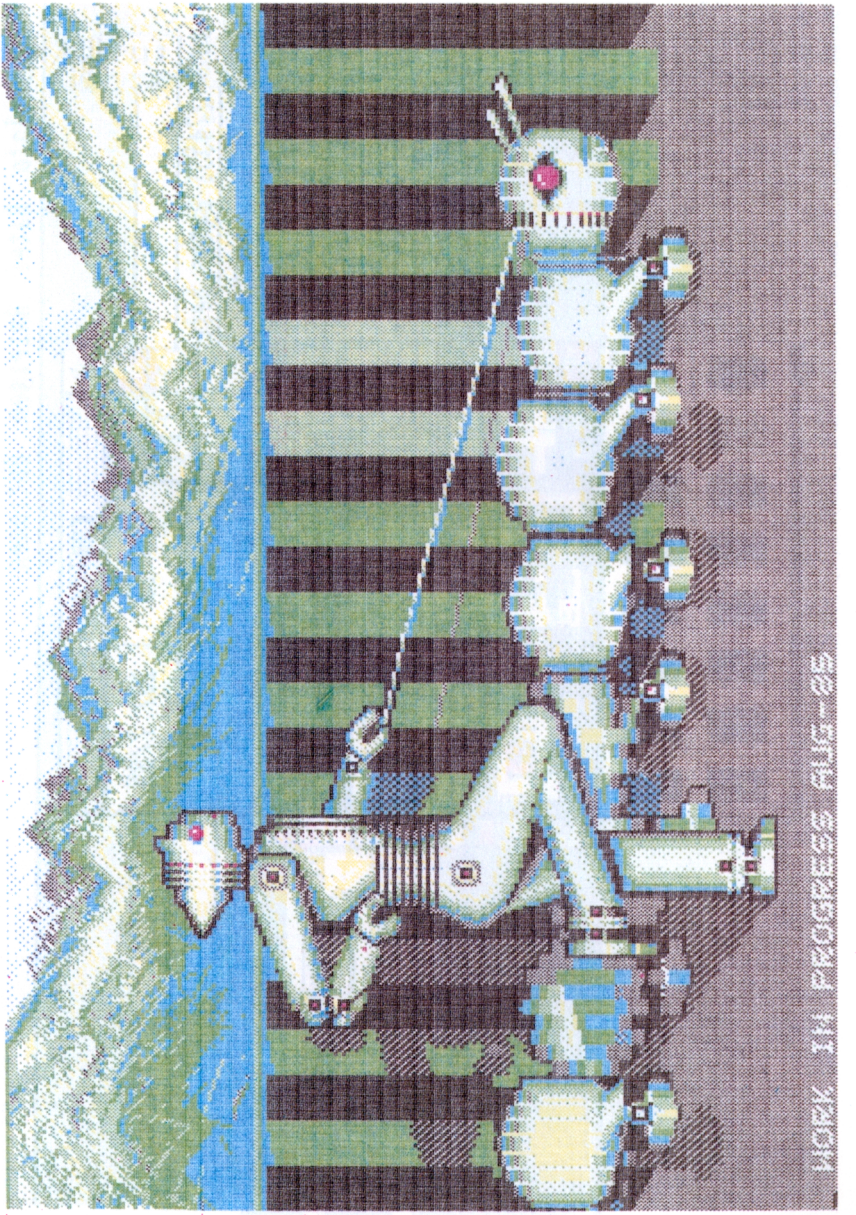


fig. 3.3.1-7

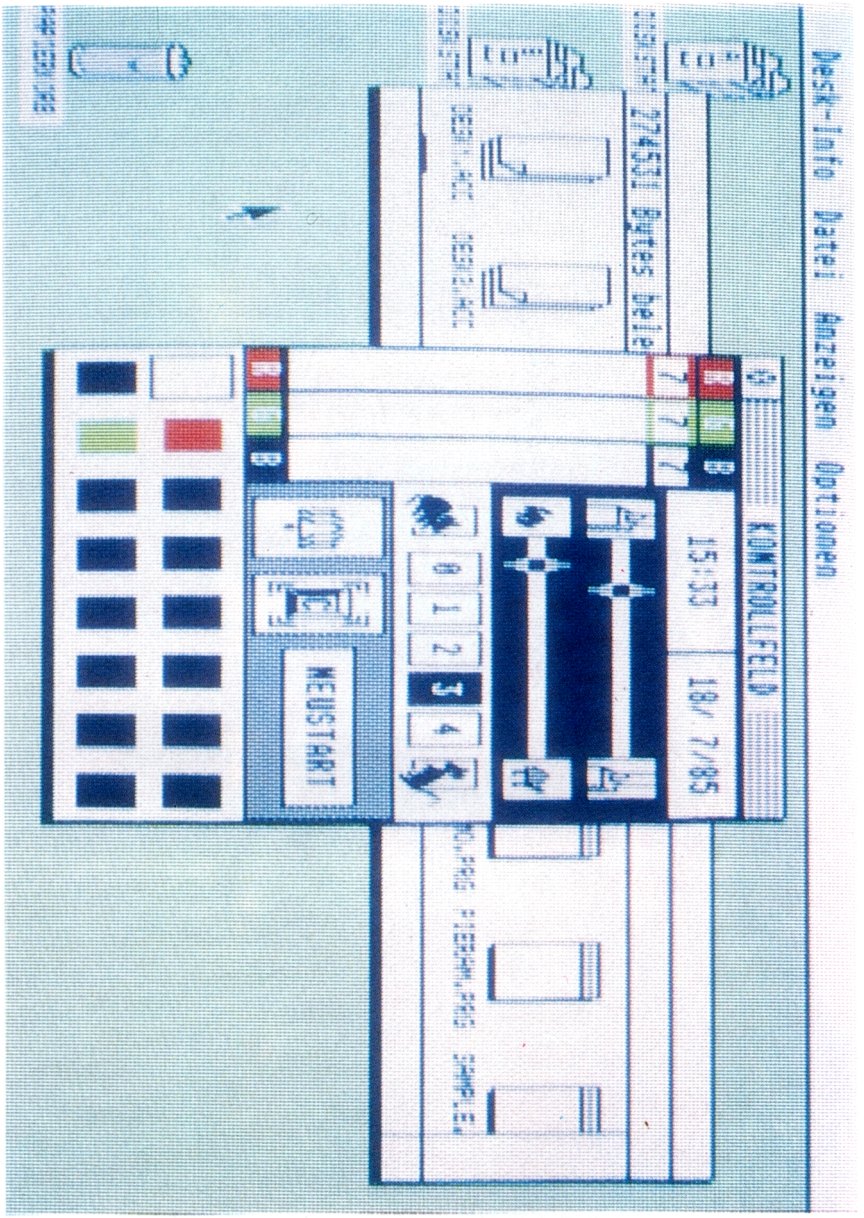

Desk-Info Datei Anzeigen Optionen
KONTROLLFELD
15:33
18/ 7/85
MEUSTART

fig. 3.3.2-5

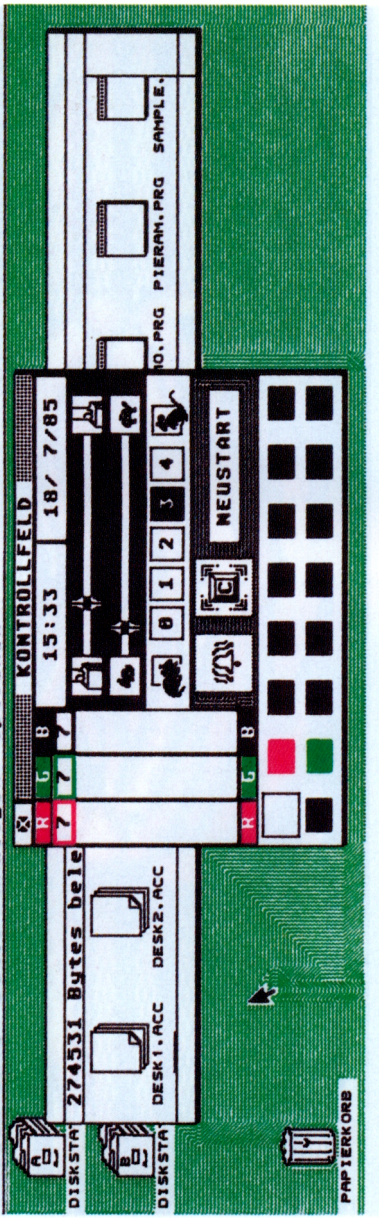

Desk-Info Datei Anzeigen Optionen
KONTROLLFELD
15:33
18/ 7/85
274531 Bytes bele
DESK1.ACC
DESK2.ACC
NEUSTART
SAMPLE.
PIERAM.PRG
DISKSTA
DISKSTA
PAPIERKORB

fig. 3.3.2-6

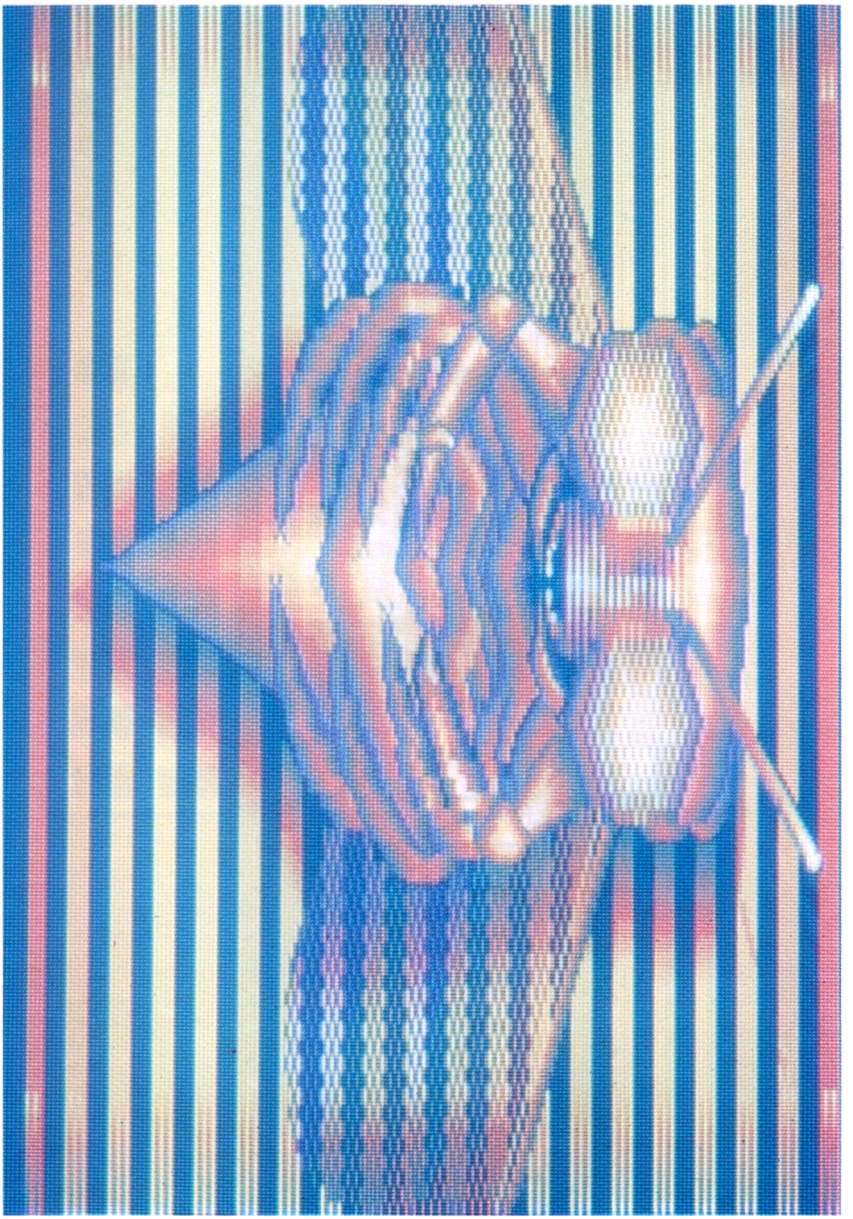


fig. 3.3.2-7

```
lf:
        andi.l  #$ffff,d7
        subq    #1,d7
lf0     move.b  0(a5,d7),d0
        bsr     chout
        dbra    d7,lf0
        rts


*************************************************************************
*                                                                       *
*       envoyer caractère en d0 vers imprimante                         *
*                                                                       *
*************************************************************************

chout:
        move.w  d0,-(a7)
        move.w  #prchar,-(a7)
        trap    #gemdos
        addq.l  #4,a7
        rts


*************************************************************************
*                                                                       *
*       constantes                                                      *
*                                                                       *
*************************************************************************

aft     dc.w    15,3,1          nbre de couleurs
pwt     dc.w    4,2,1           mots correspondants à un pixel
hmt     dc.w    2,1,1           doubler en  horizontal
vmt     dc.w    2,2,1           doubler en  vertical
zwt     dc.w    20,40,40        mots/ligne
znt     dc.w    4,4,8           aiguilles/ligne
bat     dc.w    160,160,80      position verticale de la ligne
mask    dc.l    $44001100       couleur faible
        dc.l    $aa55aa55       moyenne
```

```
        dc.l    -1                  forte
ct      dc.b    0,2,6,2,1,3,4,0     couleurs de l'imprimante
lftab   dc.b    24,"J",27,13        déplacement de ligne de 8 aiguilles
pre1    dc.b    "r",27,13           choix de couleur
pre2    dc.b    2,128,4,"*",27      mode graphique et compteur de points
fin     equ     *
        .end
```

Vous aurez la possibilité d'adapter le programme à une autre imprimante en lignes 375 à 378. Vous pouvez y inscrire la séquence de contrôle spécifique à votre imprimante, mais en sens inverse. En effet, la chaîne est sortie à partir du dernier octet (le compteur est égal au pointeur).

Si vous ne possédez pas d'assembleur et que vous voulez directement faire les adaptations dans le programme en BASIC, faites attention à ne pas modifier la longueur et l'état de la chaîne, de manière à ce qu'elle corresponde aux adresses de référence.

Le programme en langage machine du chargeur BASIC est légèrement différent de la version en assembleur. Etant donné que le programme est appelé par un CALL, il faut le terminer par RTS et non par TERM avec le GEMDOS.

```
10    dim a%(415)

20    for i=0 to 415
30    read a%(i)
40    next i
50    b=varptr(a%(0))
60    call b
70    end
950   data &H42A7,&H3F3C,&H0020
960   data &H4E41,&H5C8F,&H2C00,&H3F3C,&H0002,&H4E4E,&H548F,&H2040
970   data &HD0FC,&H7D00,&H45D0,&H43FA,&H0028,&H203C,&H0000,&H02F9
980   data &H10D9,&H51C8,&HFFFC,&H2079,&H0000,&H0456,&HD0FC,&H001C
990   data &H208A,&H2F06,&H3F3C,&H0020,&H4E41,&H5C8F,&H4E75,&H4E41
1000  data &H4A79,&H0000,&H04EE,&H6702,&H4E75,&H4E56,&HFFBE,&H3F3C
1010  data &H0002,&H4E4E,&H548F,&H2D40,&HFFF2,&H3F3C,&H0004,&H4E4E
1020  data &H548F,&HE348,&H3C80,&H43FA,&H0288,&H3D71,&H0000,&HFFFE
1030  data &H7E01,&H7008,&H1D7C,&H0007,&HFFD0,&H422E,&HFFD1,&H0C6E
1040  data &H0001,&HFFFE,&H6700,&H009A,&H3E2E,&HFFFE,&H3F3C,&HFFFF
1050  data &H3F07,&H3F3C,&H0007,&H4E4E,&H5C8F,&H4244,&H4236,&H70C0
1060  data &H3200,&H7A02,&HE949,&HE809,&H8801,&HE849,&H51CD,&HFFF8
1070  data &H1D84,&H70C0,&H0C04,&H0001,&H633A,&H7C02,&H3A3C,&H0444
1080  data &H3605,&HC640,&H6606,&HE24D,&H51CE,&HFFF6,&H7802,&H4245
1090  data &H0243,&H07FF,&H0C43,&H00FF,&H6302,&H09C5,&HE94B,&H51CC
1100  data &HFFF0,&H0C05,&H0007,&H660E,&H0C36,&H0005,&H70C0,&H6206
1110  data &H5436,&H70C0,&H4245,&H1D85,&H70D0,&H0C05,&H0006,&H6604
1120  data &H5536,&H70C0,&H51CF,&HFF86,&H7E0F,&H7008,&H0C36,&H0003
1130  data &H70C0,&H630C,&HE248,&H0C36,&H0006,&H70C0,&H6302,&H4240
1140  data &H1D80,&H70C0,&H51CF,&HFFE4,&H3016,&H43FA,&H01CA,&H3D71
1150  data &H0000,&HFFFA,&H43FA,&H01C6,&H3D71,&H0000,&HFFF8,&H43FA
1160  data &H01C2,&H3D71,&H0000,&HFFF6,&H43FA,&H01BE,&H3D71,&H0000
1170  data &HFFF0,&H43FA,&H01BA,&H3D71,&H0000,&HFFEC,&H43FA,&H01B6
1180  data &H3D71,&H0000,&HFFE8,&H3D7C,&H0032,&HFFE6,&H422E,&HFFE0
1190  data &H6016,&H536E,&HFFE6,&H6700,&H014C,&H2E2E,&HFFF2,&H0687
1200  data &H0000,&H0280,&H2D47,&HFFF2,&H4BFA,&H01A4,&H7E04,&H6100
```

```
1210  data &H0140,&H3D6E,&HFFFE,&HFFFC,&H266E,&HFFF2,&H6054,&H4A79

1220  data &H0000,&H04EE,&H6600,&H011E,&H536E,&HFFFC,&H6BC4,&H6042
1230  data &H086E,&H0001,&HFFE0,&H66E6,&H082E,&H0000,&HFFE0,&H67F0
1240  data &H43EE,&HFFD0,&HD2EE,&HFFFC,&H4246,&H1C11,&H0C06,&H0007
1250  data &H67DE,&H4BFA,&H015E,&H7E03,&H6100,&H00F6,&H43FA,&H0148
1260  data &H1031,&H6000,&H6100,&H00FE,&H4BFA,&H014B,&H7E05,&H6100
1270  data &H00E0,&H3D6E,&HFFF0,&HFFEE,&H08AE,&H0000,&HFFE0,&H49F9
1280  data &H0000,&H0000,&H3E2E,&HFFFC,&H4240,&H1036,&H70C0,&H41FA
1290  data &H010A,&H2430,&H0000,&H6012,&H536E,&HFFEE,&H6792,&H286E
1300  data &HFFE2,&HD8EE,&HFFFA,&HD8EE,&HFFFA,&H3A3C,&H8000,&H2D4C
1310  data &HFFE2,&H6004,&HE24D,&H67E0,&H3D6E,&HFFEC,&HFFEA,&H4204
1320  data &H286E,&HFFE2,&H6044,&H4287,&H3E2E,&HFFF6,&H5347,&HE30C
1330  data &H882E,&HFFE1,&H51CF,&HFFF8,&HD8EE,&HFFE8,&H536E,&HFFEA
1340  data &H6628,&H4A04,&H6706,&H08EE,&H0000,&HFFE0,&H082E,&H0001
1350  data &HFFE0,&H67C0,&H4287,&H3E2E,&HFFF8,&H5347,&H1004,&HC002
1360  data &H6162,&HE09A,&H51CF,&HFFF6,&H60AA,&H4243,&H4286,&H3C2E
1370  data &HFFFA,&H3006,&HE348,&H5306,&H41F3,&HC000,&H4BF0,&H0000
1380  data &HE30B,&H558D,&H3E15,&HCE45,&H6704,&H08C3,&H0000,&H51CE
1390  data &HFFF0,&H422E,&HFFE1,&HB66E,&HFFFC,&H6606,&H08EE,&H0000
1400  data &HFFE1,&H6082,&H4E5E,&H33FC,&HFFFF,&H0000,&H04EE,&H4E75
1410  data &H0287,&H0000,&HFFFF,&H5347,&H1035,&H7000,&H6106,&H51CF
1420  data &HFFF8,&H4E75,&H3F00,&H3F3C,&H0005,&H4E41,&H588F,&H4E75
1430  data &H000F,&H0003,&H0001,&H0004,&H0002,&H0001,&H0002,&H0001
1440  data &H0001,&H0002,&H0002,&H0001,&H0014,&H0028,&H0028,&H0004
1450  data &H0004,&H0008,&H00A0,&H00A0,&H0050,&H4400,&H1100,&HAA55
1460  data &HAA55,&HFFFF,&HFFFF,&H0002,&H0602,&H0103,&H0400,&H184A
1470  data &H1B0D,&H721B,&H0D02,&H8004,&H2A1B
```

3.3.2. TRACEUR

Nous nous trouvons face à une situation tout autre quand il s'agit d'imprimer l'écran avec un traceur.
Il serait certainement possible d'imprimer point par point comme avec l'imprimante, mais auquel cas, quel serait l'intérêt du traceur ? Un traceur est construit pour tracer des lignes. Comment écrire un programme qui trace des lignes?

Nous reconnaissons une ligne à l'écran grâce à notre représentation de l'espace, mais un programme l'analysera comme une suite de points. C'est là que se trouve le truc. Il faudra reconnaître des points reliés ensemble et les dessiner en tant que ligne ou que courbe.

Lorsqu'un point est rencontré par le programme, le crayon est abaissé et on effectue une rotation d'un quart de tour vers la gauche afin de déterminer si le point qui s'y trouve est allumé. Si ce n'est pas le cas, on effectue des rotations vers la droite en pas d'un huitième afin de rechercher des points dans cette direction. Si on en trouve un, le traceur dessine une ligne vers ce point et le processus reprend du début. On continue jusqu'à ce que l'on ne trouve plus de point dans le secteur examiné. Ce procédé apparemment absurde a pour effet de permettre de dessiner le contour de gros objets. C'est important pour l'aspect final de l'image. La figure incomplète 3.3.2-2 vous montre cet effet très clairement.

Nous ne vous cacherons pas que ce processus a un inconvénient : si un point a été rencontré, il sera effacé de l'écran afin de ne pas être retracé lors d'une recherche suivante. Le processus est destructif. Vous pourrez ainsi suivre son déroulement à l'écran.

Lorsque la recopie est finie, l'écran est blanc. Vous devrez donc faire attention à sauvegarder l'image avant de l'imprimer. Le programme a été écrit pour un traceur EPSON HI-80. On peut l'adapter sans problème à d'autres traceurs, car les commandes sont entièrement paramétrées.

L'utilisation du programme est améliorée par rapport à celui de la HARCOPY par imprimante. Afin que vous puissiez changer de stylo pour avoir plus de quatre couleurs, le traceur s'arrêtera en appuyant sur la touche ALT/HELP. Il se remettra en route après un nouvel appui sur ALT/HELP. Cela fonctionne très bien comme vous pouvez le constater à la figure 3.3.2-7. Comparez cette figure avec la 3.3.1-5. Vous voyez clairement les limites d'un traceur avec de nombreuses couleurs.
Contrairement à l'imprimante, la couleur de fond n'est pas prise en compte, Elle reste blanche.
Vous pourrez arrêter la couleur courante en appuyant trois fois sur ALT/HELP.

Une particularité de l'impression par traceur est donnée en figure 3.3.2-6. En mode 1, la résclution verticale est réduite car un point est représenté par deux point à l'écran. Mais étant donné que la routine ne double pas les points comme celle de l'imprimante, l'image est aplatie. Voyez les explications dans le listing en assembleur.

Vous trouverez le listing après les figures qui suivent. Il est suivi de quelques explications.

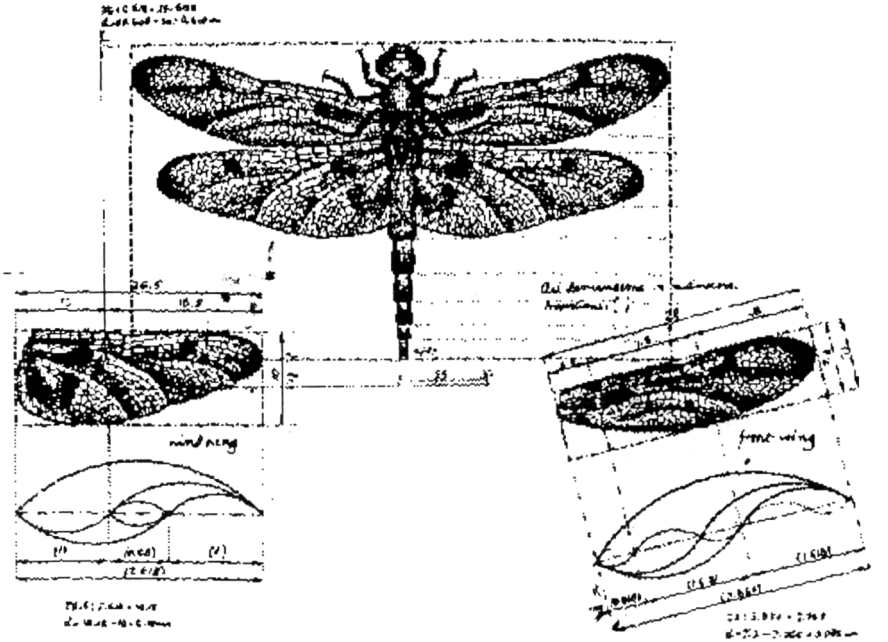


fig. 3.3.2-1

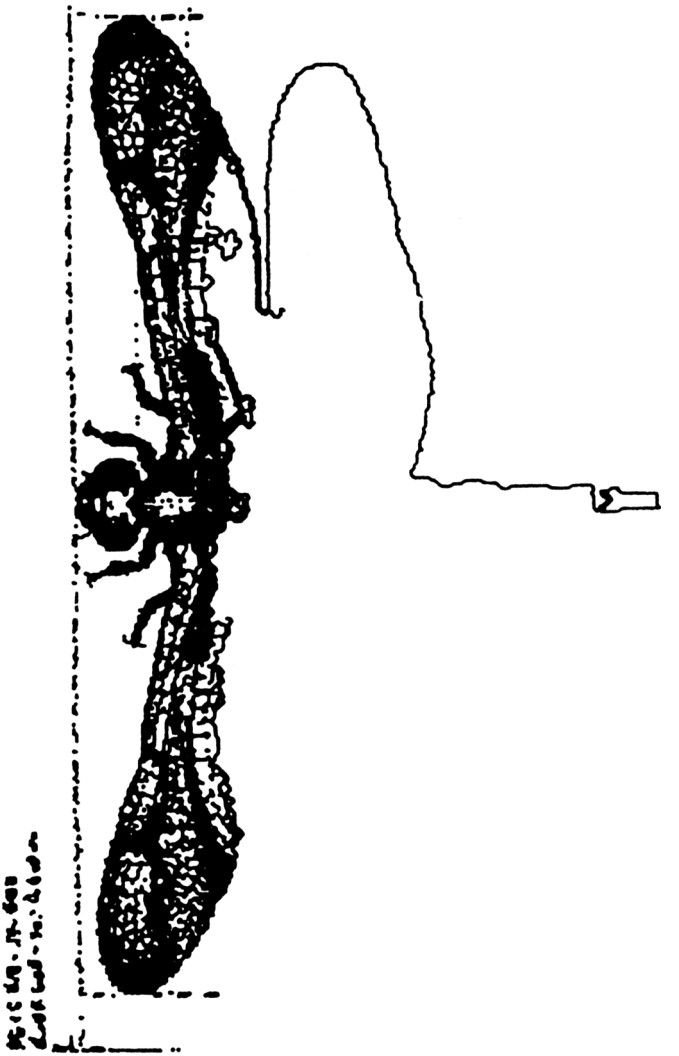


fig. 3.3.2-2

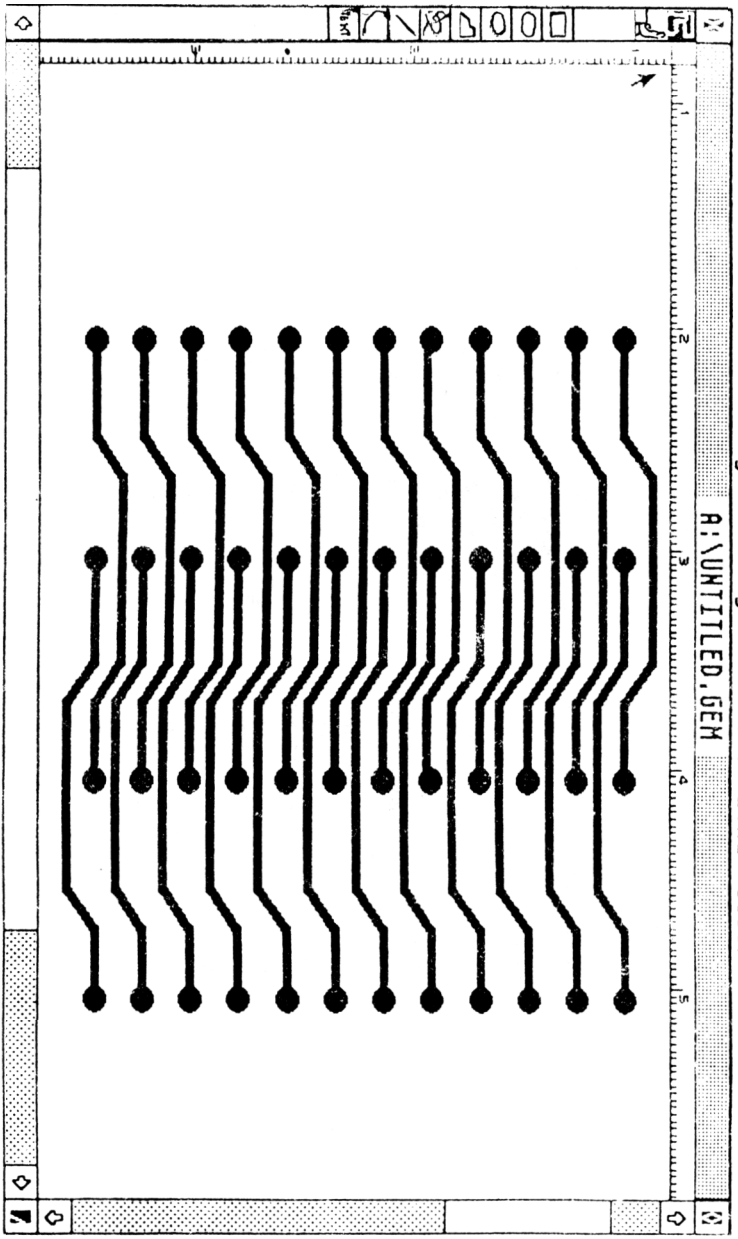
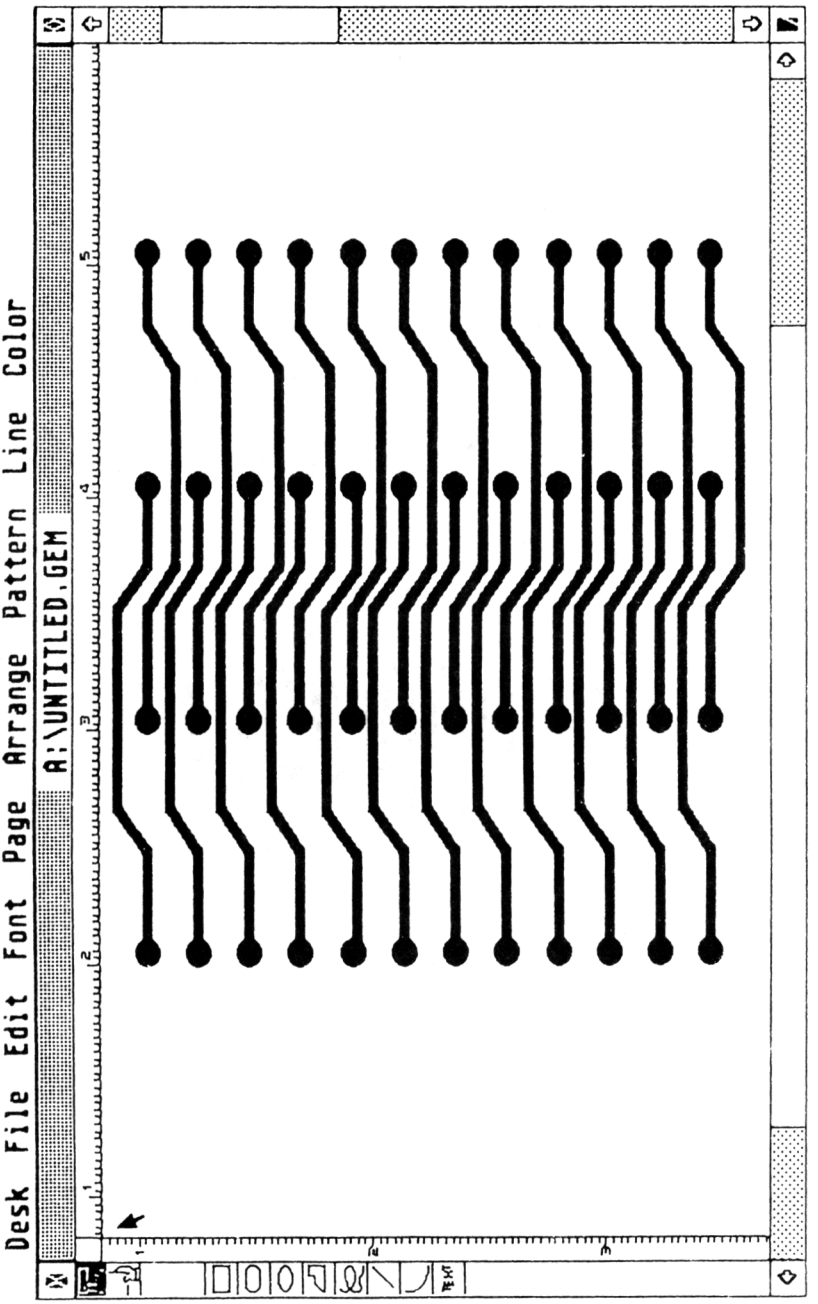

Desk File Edit Font Page Arrange Pattern Line Color
A:\UNTITLED.GEM

fig. 3.3.2-3


Desk File Edit Font Page Arrange Pattern Line Color
A:\UNTITLED.GEM

```
*       org     $cba

gemdos  equ     1
bios    equ     13
xbios   equ     14
bconout equ     3
prt     equ     0
phybas  equ     2
setscr  equ     5
super   equ     32
intin   equ     8
ptsin   equ     12
wrmod   equ     36
init    equ     $a000
setpix  equ     $a001
getpix  equ     $a002
yko     equ     2
pflag   equ     $4ee            flag alt/help
apix    equ     -4              nbre total de pixels
pscalx  equ     -6              facteurx
pscaly  equ     -8              facteury
adir    equ     -10             direction suivie
pdir    equ     -12             direction précédente
maxx    equ     -14             nbre de pixels en x
maxy    equ     -16             nbre de pixels en y
ccol    equ     -18             nr de couleur recherché
acol    equ     -20             nbre de couleurs
komma   equ     -22             virgule pour draw


*******************************************************************************
*                                                                             *
*       mettre le programme derrière la ram vidéo                             *
*                                                                             *
*******************************************************************************

dummy       lea     dummy,a0        dummy pour chargement quelconque
            clr.l   -(a7)           on a accès
```

```
          move.w  #super,-(a7)    à un

          trap    #gemdos         secteur
          addq.l  #6,a7           privilégié
          move.l  d0,d6
          move.w  #phybas,-(a7)   le début du programme
          trap    #xbios          égale
          addq.l  #2,a7           adresse de base de ram vidéo
          movea.l d0,a0           +
          adda.w  #$7d00,a0       longueur de
          lea     (a0),a2         video-ram
          lea     start(pc),a1
          move.l  #fin-start-1,d0 charger compteur
reloc     move.b  (a1)+,(a0)+     déplacer programme
          dbra    d0,reloc

          movea.l $456,a0         mettre programme dans
          adda    #28,a0          la  vblank-queue
          move.l  a2,(a0)
          move.l  d6,-(a7)        mise en
          move    #super,-(a7)    place du
          trap    #gemdos         statut
          addq.l  #6,a7           privilégié
*         rts                     si appel du basic
          clr     -(a7)
          trap    #gemdos         terminate
start:
          tst     pflag           hardcopy désirée?
          beq     param           oui>
          rts


*****************************************************************************
*                                                                           *
*       initialisation des paramètres                                       *
*                                                                           *
*****************************************************************************
```

```
param   move    #phybas,-(a7)   prendre adresse physique

        trap    #xbios          de base
        addq.l  #2,a7           de l'écran
        move    #-1,-(a7)       et
        move.l  d0,-(a7)        comparer
        move.l  d0,-(a7)        à
        move    #setscr,-(a7)   adresse
        trap    #xbios          logique
        adda.l  #12,a7
        dc.w    init            prendre paramètres de lécran
        link    a6,#-24         faire de la place pour secteur de travail
        movea.l intin(a0),a3
        movea.l ptsin(a0),a4
        clr     wrmod(a0)       mettre en mode écriture
        move    (a0),d7         nbre de tracés
        andi    #6,d7
        lea     scalx(pc),a0
        move    0(a0,d7),pscalx(a6)
        lea     scaly(pc),a0
        move    0(a0,d7),pscaly(a6)
        lea     max(pc),a0
        move    0(a0,d7),maxx(a6)
        lea     may(pc),a0
        move    0(a0,d7),maxy(a6)
        lea     colc(pc),a0
        move    0(a0,d7),acol(a6)
        move    maxx(a6),d6
        mulu    maxy(a6),d6
        move.l  d6,apix(a6)
        move    #1,ccol(a6)     nr de couleur. 1
init1   cmpi    #1,pflag        stop avant choix de couleur?
        bne     init3           non  >
        bsr     caps
init2   cmpi    #2,pflag        continuer?
        bne     init2           non  >
        clr     pflag
init3   bsr     setcol          choix de couleur
```

```
          bsr     home            traceur en mode impression

          pea     -1              la recherche commence en haut à gauche


*****************************************************************************
*                                                                           *
*       rechercher premier pixel d'une ligne                                *
*                                                                           *
*****************************************************************************

such      move.l  (a7)+,d7
          addq.l  #1,d7
          cmp.l   apix(a6),d7     tous les pixels ont-ils été testés
          beq     exit            oui>
          move.l  d7,-(a7)        sauvegarder position courante
          bsr     chkpix          rechercher point suivant
          cmp     ccol(a6),d0     couleur rcherchée?
          bne     such            non  >
          move    #3,adir(a6)     la direction de recherche est à droite


*****************************************************************************
*                                                                           *
*       imprimer points reliés ensembles                                    *
*                                                                           *
*****************************************************************************

plot      bsr     mov             traceur sur nouvelle position
          bsr     pendwn          abaisser stylo
          bsr     erase           effacer point trouvé
plot1     clr     d0
          bsr     nexpix          rechercher point relié
          tst     d0              couleur convenable trouvée?
          bne     plot2           oui>
          bsr     outcr           imprimer délimiteur
          bsr     penup           relever stylo
          bra     such
```

```
plot2     bsr     draw            ligne vers le points suivant

          bsr     erase           effacer les points imprimés
          bra     plot1           recherche du point suivant


*****************************************************************************
*                                                                           *
*       recherche des points reliés suivants                                *
*                                                                           *
*****************************************************************************

nexpix    subq    #2,adir(a6)     1/4 de tour à gauche
          andi    #7,adir(a6)     seulement 0-7 autorisés
          move    adir(a6),pdir(a6) mémoriser direction d'impression
          bra     nex3
nex1      movem   (a7)+,d3-d4     prendre anciennes coordonnées
          addq    #1,adir(a6)     1/8 de tour à droite
          andi    #7,adir(a6)     seulement 0-7 autorisés
          move    pdir(a6),d7
          cmp     adir(a6),d7     retour au point imprimé ?
          bne     nex3            non  >
          clr     d0
          rts
nex3      move    adir(a6),d7     préparer
          lsl     #1,d7           un branchement
          lea     j(pc),a0        indépendant de
          adda    0(a0,d7),a0     la direction
          movem   d3-d4,-(a7)     sauvegarder coordonnées
          jsr     (a0)            retour
          cmp     ccol(a6),d0     couleur convenable trouvée?
          bne     nex1            non  > essayer une autre direction
          addq.l  #4,a7           corriger la pile
          rts                     relier les points
```

```
*****************************************************************************
*                                                                           *
*       branchements dépendant de la direction                              *
*                                                                           *
*****************************************************************************

j         dc.w    re-j,ru-j,un-j,lu-j,li-j,lo-j,ob-j,ro-j

re        addq    #1,d3           à droite
          cmp     maxx(a6),d3     fin de la ligne est-elle atteinte?
          bcs     askpix          non  >
          rts

ru        addq    #1,d3           en bas à droite
          cmp     maxx(a6),d3     finde la ligne?
          bcs     un              non  >
          rts

un        addq    #1,d4           vers le bas
          cmp     maxy(a6),d4     fin de l'écran??
          bcs     askpix          non  >
          rts

lu        addq    #1,d4           en bas à gauche
          cmp     maxy(a6),d4     fin de l'écran ?
          bcs     li              non  >
          rts

li        subq    #1,d3           à gauche
          bpl     askpix          pas encore la fin
          rts

lo        subq    #1,d3           en haut à gauche
          bpl     ob              pas encore la fin
          rts
```

```
ob      subq    #1,d4           en haut

        bpl     askpix          pas encore la fin
        rts

ro      subq    #1,d4           en ahut à droite
        bpl     re              pas encore la fin
        rts


*****************************************************************************
*                                                                           *
*       tester les pixels allumés                                           *
*                                                                           *
*****************************************************************************

chkpix  divu    maxx(a6),d7     modifier d7 d'après
        move    d7,d4           y et
        swap    d7              x
        move    d7,d3
askpix  cmpi    #3,pflag        coupure de couleur?
        bcs     ask1            non >
        move    #1,pflag        autoriser un changement de stylo éventuel
        bra     exit            couleur prète>
ask1    move    d3,(a4)         charger
        move    d4,yko(a4)      coordonnées
        dc.w    getpix          test du point line a
        rts


*****************************************************************************
*                                                                           *
*       effacer pixel                                                       *
*                                                                           *
*****************************************************************************

erase   move    d3,(a4)         charger
        move    d4,yko(a4)      coordonnées
```

```
          clr     (a3)            couleur 0

          dc.w    setpix          afficher point line a
          rts


*******************************************************************************
*                                                                             *
*       routines de sortie diverses                                           *
*                                                                             *
*******************************************************************************

home      lea     hm(pc),a2       traceur en mode de base
          bra     outstrx

setcol    lea     scols(pc),a2    fixer couleur d'après ccol
          bsr     outstrx
          lea     scoln(pc),a2
          move    ccol(a6),d7
          move.b  -1(a2,d7),d0
          bsr     outchr
          bra     outcr

penup     lea     pup(pc),a2      relever crayon
          bra     outstrx

pendwn    clr     komma(a6)       abaisser crayon
          lea     pdw(pc),a2
          bra     outstrx

mov       lea     mv(pc),a2       positionnement sans crayon
          bsr     outstrx
          bsr     outkor
          bra     outcr

draw      tst     komma(a6)       positionnement avec crayon
          bne     draw1
          st      komma(a6)
```

```
          lea     dr(pc),a2

          bsr     outstrx
          bra     outkor
draw1     bsr     outkom

outkor    move    d3,d6           impression de coordonnées en ascii
          mulu    pscalx(a6),d6
          bsr     outw
          bsr     outkom          afficher virgule
          move    maxy(a6),d7     inverser
          sub     d4,d7           coordonnée y
          move    d7,d6
          mulu    pscaly(a6),d6

outw      move.l  #1000,d7        imprimer mot hexa en d6 sous forme ascii
outw1     andi.l  #$3fff,d6
          divu    d7,d6
outw3     move    d6,d0
          ori     #48,d0
          bsr     outchr
          swap    d6
outw4     divu    #10,d7
          bne     outw1
          rts

outstrx   clr     d2              imprimer chaine (compteur 1 en (a2))
          move.b  (a2)+,d2
outstr    move.b  (a2)+,d0        imprimer chain en (a2) (compteur en (d2))
          bsr     outchr
          dbra    d2,outstr
          rts

caps      lea     cap(pc),a2
          bsr     outstrx

outcr     move    #13,d0          imprimer cr
          bra     outchr
```

```
outkom      move    #44,d0          imprimer virgule


outchr      movem.l d0-d2/a0-a2,-(a7)
            andi    #255,d0
            move    d0,-(a7)        carcatère en d0
            move    #prt,-(a7)      à envoyer
            move    #bconout,-(a7)  vers imprimante
            trap    #bios
            addq.l  #6,a7
            movem.l (a7)+,d0-d2/a0-a2
            rts


*******************************************************************************
*                                                                             *
*       sortie                                                                *
*                                                                             *
*******************************************************************************

exit        addq    #1,ccol(a6)     couleur recherchée+1
            move    acol(a6),d7
            cmp     ccol(a6),d7     toutes les couleurs traitées?
            bpl     init1           non >
exitx       unlk    a6              libérer plac réservée
            move    #-1,pflag       effacer harcopy-flag
            bsr     home            traceur en mode de base
            bra     caps            chapeau sur le crayon


*******************************************************************************
*                                                                             *
*       constantes                                                            *
*                                                                             *
*******************************************************************************

scalx       dc.w    4,4,4           facteurs x
scaly       dc.w    4,4,4           facteurs y
```

```
max     dc.w    640,640,320     nbre de pixels en x

may     dc.w    400,200,200     nebre de pixels en y
colc    dc.w    1,3,15          nbre de couleurs
mv      dc.b    1,"MA"          move absolu
cap     dc.b    3,"SP-1"        échange de crayon
pup     dc.b    5,"MR0,0",13    move relatif (pen up)
pdw     dc.b    5,"DR0,0",13    draw relatif (pen down)
dr      dc.b    1,"DA"          draw absolu
hm      dc.b    3,13,"HO",13    mode de base
scols   dc.b    1,"SP"          échange de couleur
scoln   dc.b    "123412341234123"
fin     equ     *
          .end
```

Vous pouvez ici encore adapter le programme grâce aux lignes 371 à 378. Nous avons choisi un autre procédé qu'avec la harcopy par imprimante pour envoyer la chaîne. Le compteur-1 est situé dans ce cas avant le texte qui est alors envoyé en sens normal.

Les lignes 366 et 367 indiquent le nombre de pas que le traceur doit effectuer pour un point sur l'écran. On choisira le pas en fonction de l'épaisseur du stylomine. Dans notre cas c'est 0.4 mm. Le chargeur BASIC est lui aussi terminé par l'instruction RTS au lieu de TERM.

```
10    dim a%(411)

20    for i=0 to 411
30    read a%(i)
40    next i
50    b=varptr(a%(0))
60    call b
70    end
950   data &H42A7,&H3F3C,&H0020
960   data &H4E41,&H5C8F,&H2C00,&H3F3C,&H0002,&H4E4E,&H548F,&H2040
970   data &HD0FC,&H7D00,&H45D0,&H43FA,&H0028,&H203C,&H0000,&H02F1
980   data &H10D9,&H51C8,&HFFFC,&H2079,&H0000,&H0456,&HD0FC,&H001C
990   data &H208A,&H2F06,&H3F3C,&H0020,&H4E41,&H5C8F,&H4E75,&H4E41
1000  data &H4A79,&H0000,&H04EE,&H6702,&H4E75,&H3F3C,&H0002,&H4E4E
1010  data &H548F,&H3F3C,&HFFFF,&H2F00,&H2F00,&H3F3C,&H0005,&H4E4E
1020  data &HDFFC,&H0000,&H000C,&HA000,&H4E56,&HFFE8,&H2668,&H0008
1030  data &H2868,&H000C,&H4268,&H0024,&H3E10,&H0247,&H0006,&H41FA
1040  data &H0264,&H3D70,&H7000,&HFFFA,&H41FA,&H0260,&H3D70,&H7000
1050  data &HFFF8,&H41FA,&H025C,&H3D70,&H7000,&HFFF2,&H41FA,&H0258
1060  data &H3D70,&H7000,&HFFF0,&H41FA,&H0254,&H3D70,&H7000,&HFFEC
1070  data &H3C2E,&HFFF2,&HCCEE,&HFFF0,&H2D46,&HFFFC,&H3D7C,&H0001
1080  data &HFFEE,&H0C79,&H0001,&H0000,&H04EE,&H6614,&H6100,&H01CE
1090  data &H0C79,&H0002,&H0000,&H04EE,&H66F6,&H4279,&H0000,&H04EE
1100  data &H6100,&H012C,&H6100,&H0120,&H4879,&HFFFF,&HFFFF,&H2E1F
1110  data &H5287,&HBEAE,&HFFFC,&H6700,&H01CC,&H2F07,&H6100,&H00D2
1120  data &HB06E,&HFFEE,&H66E8,&H3D7C,&H0003,&HFFF6,&H6100,&H012C
1130  data &H6100,&H011E,&H6100,&H00E4,&H4240,&H6118,&H4A40,&H660A
1140  data &H6100,&H0180,&H6100,&H0104,&H60C4,&H6100,&H0118,&H6100
1150  data &H00CA,&H60E4,&H556E,&HFFF6,&H026E,&H0007,&HFFF6,&H3D6E
1160  data &HFFF6,&HFFF4,&H601C,&H4C9F,&H0018,&H526E,&HFFF6,&H026E
1170  data &H0007,&HFFF6,&H3E2E,&HFFF4,&HBE6E,&HFFF6,&H6604,&H4240
1180  data &H4E75,&H3E2E,&HFFF6,&HE34F,&H41FA,&H0016,&HD0F0,&H7000
1190  data &H48A7,&H1800,&H4E90,&HB06E,&HFFEE,&H66CA,&H588F,&H4E75
1200  data &H0010,&H001A,&H0024,&H002E,&H0038,&H003E,&H0044,&H004A
```

```
1210  data &H5243,&HB66E,&HFFF2,&H6542,&H4E75,&H5243,&HB66E,&HFFF2

1220  data &H6502,&H4E75,&H5244,&HB86E,&HFFF0,&H652E,&H4E75,&H5244
1230  data &HB86E,&HFFF0,&H6502,&H4E75,&H5343,&H6A1E,&H4E75,&H5343
1240  data &H6A02,&H4E75,&H5344,&H6A12,&H4E75,&H5344,&H6AC2,&H4E75
1250  data &H8EEE,&HFFF2,&H3807,&H4847,&H3607,&H0C79,&H0003,&H0000
1260  data &H04EE,&H650C,&H33FC,&H0001,&H0000,&H04EE,&H6000,&H00D6
1270  data &H3883,&H3944,&H0002,&HA002,&H4E75,&H3883,&H3944,&H0002
1280  data &H4253,&HA001,&H4E75,&H45FA,&H0113,&H6000,&H0082,&H45FA
1290  data &H0110,&H6100,&H007A,&H45FA,&H010B,&H3E2E,&HFFEE,&H1032
1300  data &H70FF,&H6100,&H0084,&H6000,&H007A,&H45FA,&H00DE,&H605E
1310  data &H426E,&HFFEA,&H45FA,&H00DB,&H6054,&H45FA,&H00C6,&H614E
1320  data &H6116,&H605E,&H4A6E,&HFFEA,&H660C,&H50EE,&HFFEA,&H45FA
1330  data &H00C8,&H613A,&H6002,&H614E,&H3C03,&HCCEE,&HFFFA,&H610E
1340  data &H6144,&H3E2E,&HFFF0,&H9E44,&H3C07,&HCCEE,&HFFF8,&H2E3C
1350  data &H0000,&H03E8,&H0286,&H0000,&H3FFF,&H8CC7,&H3006,&H0040
1360  data &H0030,&H6124,&H4846,&H8EFC,&H000A,&H66E8,&H4E75,&H4242
1370  data &H141A,&H101A,&H6112,&H51CA,&HFFFA,&H4E75,&H45FA,&H0067
1380  data &H61EC,&H700D,&H6002,&H702C,&H48E7,&HE0E0,&H0240,&H00FF
1390  data &H3F00,&H3F3C,&H0000,&H3F3C,&H0003,&H4E4D,&H5C8F,&H4CDF
1400  data &H0707,&H4E75,&H526E,&HFFEE,&H3E2E,&HFFEC,&HBE6E,&HFFEE
1410  data &H6A00,&HFDF0,&H4E5E,&H33FC,&HFFFF,&H0000,&H04EE,&H6100
1420  data &HFF26,&H60B8,&H0004,&H0004,&H0004,&H0004,&H0004,&H0004
1430  data &H0280,&H0280,&H0140,&H0190,&H00C8,&H00C8,&H0001,&H0003
1440  data &H000F,&H014D,&H4103,&H5350,&H2D31,&H054D,&H5230,&H2C30
1450  data &H0D05,&H4452,&H302C,&H300D,&H0144,&H4103,&H0D48,&H4F0D
1460  data &H0153,&H5031,&H3233,&H3431,&H3233,&H3431,&H3233,&H3431
1470  data &H3233
```

CHAPITRE 4

L'ENVIRONNEMENT GEM

GEM est aussi agréable à manipuler en tant qu'interface de programmation avec les fonctions de GEMDOS qu'il l'est pour l'utilisateur. Un programme sous GEM sera généralement plus facile et plus rapide à écrire qu'un programme avec un système d'exploitation courant. En effet, GEM contient de nombreux sous programmes que le programmeur peut tout simplement appeler. Il lui suffit de savoir ce dont il dispose avec GEM et comment il peut relier ces routines avec son application. Un programme sous GEM se distingue d'un programme sous un système d'exploitation comme par exemple CP/M 68K, même quand on ignore la différence subtile entre l'accessory et l'application. C'est sans doute la raison pour laquelle les programmes pour le ST se font tellement attendre : l'aspect utilisateur représente en effet une toute autre optique de programmation. Alors qu'auparavant la partie initialisation d'un programme consistait essentiellement dans la mise en place de certaines variables, le programmeur sur ATARI ST doit maintenant s'occuper avant tout de sa station de travail, c'est à dire par exemple se demander avec quelles coordonnées il va travailler et ainsi de suite. Si on a compris cette façon de procéder, si on a écrit les routines correspondantes et que l'on connait parfaitement leur fonctionnement, on peut se concentrer en développeur sur le principal et manipuler des routines de gestion des fenêtres, de test de la souris et du clavier avec le minimum de travail. En effet, ces parties d'initialisation sont utilisables pour d'autres applications sans grandes adaptations. (On peut même en éviter certaines).
Ainsi tout programmeur qui travaille sous GEM se construira sa collection de sous programmes qui constituera une bibliothèque qui lui allègera le travail de programmation car il pourra les réutiliser ensemble. Nous allons maintenant vous présenter quelques unes de ces routines standard ainsi que le développement d'accessoires et d'applications avec un programme qui vous aidera à changer la police de caractère de votre imprimante.

Afin de comprendre la structure et le mode de fonctionnement du programme, il faut avoir une vision claire du système d'exploitation de l'ATARI ST. Nous allons donc tout d'abord vous présenter en gros le GRAPHIC ENVIRONMENT MANAGER.

4.1.1. ANATOMIE DE GEM

GEM, dont la plus grande part à été écrite en C, est orienté graphisme et facilite par sa technique particulière la manipulation de l'ATARI. Un utilisateur manipulera sans problème les fonctions de GEM sans se rendre compte de la structure technique du contrôle de la souris, de la manipulation des icônes, des menus déroulants et des fenêtres. Il se moque certainement aussi des caractéristiques techniques spécifiques d'un 520 ST ou d'un 520 ST+. Combien de temps devrait passer un programmeur pour écrire des programmes capables de remplir le rôle de ces routines?

Mais pourquoi devrait-il réinventer le monde une seconde fois alors qu'il peut utiliser les routines GEM ? Aussi loin que nous les connaissions, tous les langages de programmation existant sur l'ATARI disposent des bibliothèques nécessaires pour écrire des programmes convenables. Heureusement, il apparaît également que les noms des routines sont les mêmes, si bien que les exemples de programmes qui suivent seront explicites pour les programmeurs en PASCAL, en MODULA-2 ou en C. Les codes sources, et en général les programmes qui en résultent, ne se distingueront pas essentiellement par l'utilisation des bibliothèques GEM.

En effet, GEM dispose de routines de transfert de données allant de l'affichage de chaînes de caractères à l'écran jusqu'à la gestion des masques de saisie ou d'affichage que l'on devrait plutôt appeler 'formulaires' sous GEM.

Le 'VIRTUAL INTERFACE DEVICE' et l''APPLICATION ENVIRONMENT SERVICES' sont indispensables pour ces tâches. Derrière ces noms se cachent les parties de programmes qui servent à la manipulation du matériel de l'ordinateur et qui

permettent au programmeur de le considérer comme une boite noire. Le VDI est tellement étendu qu'il autorise deux systèmes de coordonnées. Un des deux adresse un périphérique mathématiquement normé. Il existe donc des programmes qui ne tiennent pas compte de la sortie, qu'elle soit sur écran ou sur imprimante. On comprend donc le premier traitement indispensable d'une application GEM, qui est l'ouverture du canal poste de travail.Le VDI doit savoir où, et avant tout sous quelle forme, les données doivent être affichées.

Pour le programmeur, le gros avantage est qu'il ne rencontrera aucun problème pour l'adaptation d'un programme à un autre périphérique ou à un autre ordinateur fonctionnant sous GEM, car il n'y a rien de changé dans les routines d'entrée-sortie.

De plus, le système du VDI a prévu de nombreuses routines graphiques, comme des routines de dessin (polyline, Circle, Fill) et des fonctions de saisie et d'affichage.

L'AES gère les données provenant de périphériques. Il doit entre autres lire le clavier, vérifier le déplacement de la souris et contrôler les touches de la souris, ainsi que des éléments de sortie, comme les fenêtres, les icônes et les menus déroulants. En cela, l'AES est prévu pour une gestion multitâche. Pensez au spooler d'imprimante ou bien à l'horloge de la ligne contrôle qui fonctionnent en plus du programme principal.

4.1.2. LE VIRTUAL DEVICE INTERFACE

Le VDI est constitué de deux parties : le GRAPHIC DEVICE OPERATING SYSTEM (GDOS) qui contient de nombreuses routines graphiques indépendantes des périphériques et le driver de périphériques qui contient des informations sur le type de caractère utilisé, les motifs. Ainsi on doit indiquer au VDI que le moniteur noir et blanc du ST ne possède que deux couleurs et se limite aux coordonnées 0,0 à 639,399. Par contre si on utilise un moniteur couleur, le VDI peut avoir accès à 16 couleurs et il sait que la résolution de ce poste de travail n'est pas aussi haute.

On comprend enfin que si le programmeur trace une ligne horizontale de 600 points, elle ne sera dessinée qu'à moitié sur le moniteur couleur qui ne possède que 320 points par ligne. Mais afin d'être indépendant des périphériques, le VDI dispose de plusieurs systèmes de coordonnées :

-NDC, NORMALIZED DEVICE COORDINATES et
- RC, RASTER COORDINATES

Alors que les 'Raster coordinates' correspondent à l'écran physique qui est de 320*200 et 640*400 sur l'ATARI ST, et pour lequel les coordonnées x et y sont modifiées, les coordonnées 'normalisées' proposent une image idéale. Leur principe correspond mieux à notre système de coordonnées cartésiennes habituel : l'origine se situe en bas à gauche et que le point dont les coordonnées sont les plus grandes est situé en haut à droite.

Le secteur des coordonnées adressables est 0,0 à 32767,32767, ce qui correspond à une image géométriquement correcte dont la résolution est très haute. (ou bien de très grande taille).
Le programmeur sous GEM doit choisir le système de coordonnées qu'il utilisera.

S'il choisit le NDC, les valeurs données durant le fonctionnement de GDOS (la partie de GEM qui est utilisée pour cela) seront recalculées en fonction des coordonnées réelles. Si on dessine un carré de 100 de coté, il apparaît comme un carré à l'écran. Si le programmeur se décide au contraire pour les 'raster coordinates', le VDI ne fait aucun calcul. Le programmeur devra donc faire ses propres adaptations en fonction de son programme.
Le système NDC utilisé avec le VDI est utile essentiellement pour l'échange de graphisme entre différents périphériques. Ainsi, nous rencontrerons toujours les mêmes difficultés avec les coordonnées 'raster' et le contrôle des imprimantes, en particulier en ce qui concerne le rapport entre la longueur et la largeur. En effet, avec un système de coordonnées de 600*200, la relation entre les coordonnées à l'écran est environ de 1:1.8.

Ainsi, un carré n'apparaîtra pas comme un carré à l'imprimante, à moins qu'il ne soit imprimé par chance avec le même système de coordonnées.

Mais dans le cas ou le VDI est adressé, le driver de périphériques peut prendre en charge les recalculs nécessaires. Ainsi, les images apparaîtront sur les périphériques telles qu'elles sont sur l'écran. L'inconvénient peut être le temps de traitement de tels programmes qui est plus élevé. Un graphisme doit être d'abord recalculé dans le mode courant avant d'être affiché. Pour cette raison, il est préférable de travailler avec des coordonnées 'raster' à moins que vous ne deviez absolument écrire un programme portable.

4.1.3. L'APPLICATION ENVIRONMENT SERVICES

L'AES est constitué de plusieurs parties:

- la bibliothèque de sous programmes
- le dispatcher
- le shell
- le desk accessory buffer
- le tampon de menu et d'alarme

Le tampon de menu et d'alarme est la fonction la plus rapide de GEM. Ainsi, le tampon de menu et d'alarme mémorisera temporairement la partie de l'écran qui sera recouverte par un menu déroulant. Après l'utilisation du menu en question, un sous programme se charge de restaurer très rapidement l'écran. Il n'est pas nécessaire que l'application, ou le programmeur, s'occupe de cela. Il faut remarquer que seule la place pour un quart d'écran est réservée. Pour cette raison, ce n'est pas la peine d'essayer d'étendre un menu sur l'écran tout entier!

Le desk accessory buffer auquel nous nous intéresserons plus tard est construit selon le même principe. En l'occurence, on sauvegarde non seulement les données dans cette place mémoire réservée mais aussi des programmes d'aide tel PR.INIT, que le programmeur désire appeler durant le déroulement d'un programme.

Le travail du dispatcher est de faire correspondre des travaux que le processeur doit exécuter en même temps. 'en même temps' est une notion relative car ce qui nous paraît simultané est en fait exécuté très rapidement consécutivement par le CPU à 8 Mhz.
Pour ne pas mélanger les temps d'exécution, le dispatcher possède deux listes : la 'ready list' dans laquelle sont exécutés successivement les programmes qui tournent déjà et la 'not ready list' dans laquelle se trouvent les processus qui attendent un résultat précis avant d'être exécutés. Un tel résultat est par exemple :

- presser sur une touche
- presser un bouton de la souris
- le déplacement de la souris
- une transmission
- ou laisser s'écouler un temps déterminé

Ainsi, notre programme d'adaptation de l'imprimante est toujours dans la 'not ready list' et attend que le desk accessory FXINIT soit appelé.
Si c'est le cas, l'utilisateur aura pointé avec sa souris sur la ligne correspondante dans le menu DESK et pressé le bouton gauche de la souris. Le programme est alors effacé de la 'not ready list' et mis dans la 'ready list'.

Le processus décrit plus haut y est suivi et, après un certain temps, mis à la fin de la liste si bien que le processus suivant peut être exécuté. De cette façon, le dispatcher divise le temps CPU entre l'application en cours, comme par exemple le SPOOLER d'imprimante, et les routines du système d'exploitation.
En tout, le dispatcher peut gérer jusqu'à six tâches en plus de ces appels de l'AES. Il peut s'agir des programmes se trouvant dans le desk accessory code buffer.

Il est important de bien saisir la différence avec la pratique courante pour comprendre les programmes GEM :
Si vous avez déjà programmé auparavant, vous avez sans doute prévu une boucle dans votre programme qui teste si un résultat est obtenu (INKEY$).

Ou bien vous avez tout simplement stoppé le déroulement du programme (INPUT). tout simplement : vous dites à l'ordinateur d'attendre que telle ou telle opération soit réalisée.

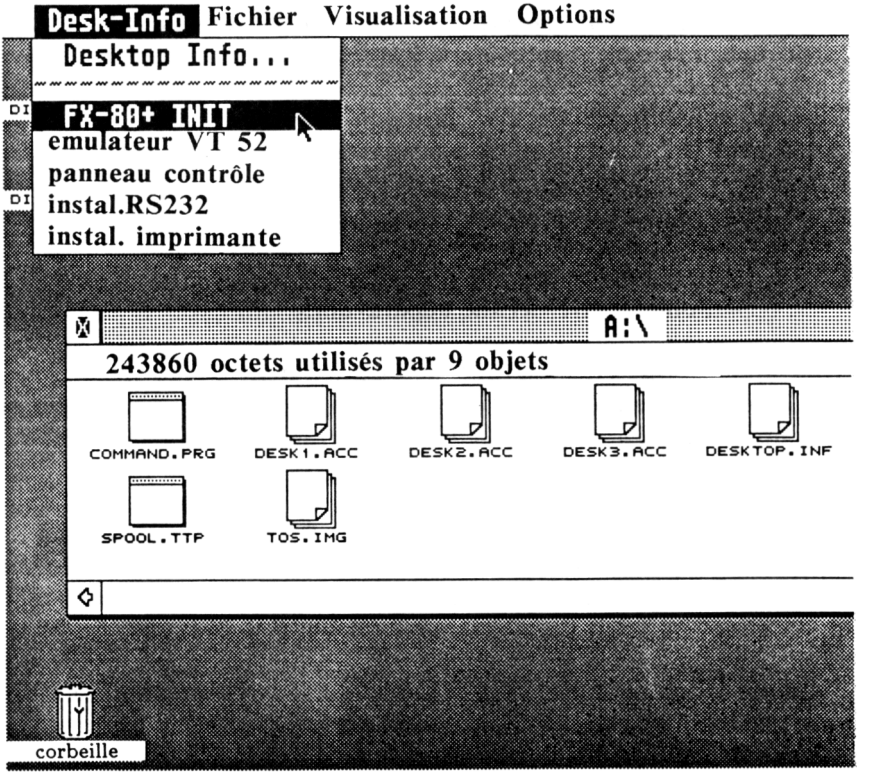
Sur l'ATARI au contraire, vous pouvez indiquer au système qu'il peut exécuter autre chose tant que le résultat attendu n'est pas obtenu. Jusqu'à ce moment, le programme reste sur la 'not ready list' et on ne perd pas de temps CPU en attente inutile. C'est ce processus qui rend possible le multitâche car les routines de la 'ready list' sont exécutées.

La bibliothèque contient certains sous programmes pour la manipulation des fenêtres, le test de la souris, l'affichage de messages système comme les 'dialogbox' et les menus déroulants. Même la routine de test des dialogbox (OK, CANCEL, RETRY) et les fonctions de contrôle de la fenêtre se trouvent dans cette partie. Evidemment, le programmeur sous GEM peut et doit se servir de ces routines. Entre autres, la bibliothèque de L'AES dispose de routines qui initialisent les programmes à mettre en route, c'est à dire des routines qui font en sorte que le programme d'application ait la priorité sur toutes les autres tâches. Le programmeur doit bien sûr là encore indiquer si le programme doit être traité comme une application ou bien comme un accessoire.

Le screen manager s'occupe du contrôle de la souris dès qu'elle se trouve hors de la zone de travail de la fenêtre active. La zone de travail est constituée du contenu de la fenêtre. Le titre et la ligne d'information n'en font pas partie.
Le screen manager est activé aussitôt que l'utilisateur quitte la fenêtre supérieure, c'est à dire par exemple quand il utilise la liste des choix avec les menus déroulants.
Il prend en compte la manipulation de l'utilisateur et l'exécute, c'est à dire qu'il indique à l'application courante qu'il faut redessiner la fenêtre.

Le SHELL enfin est une partie de la bibliothèque de l'AES. Elle se trouve au début de la ready list lors de l'Appel du desktop.

Il s'occupe de l'appel d'une application et il est toujours situé entre l'application et le desktop. Ainsi à la mise en route du programme, le desktop informe le SHELL s'il s'agit d'une application TOS ou GEM ainsi que du nom du sous directory correct (il s'agit de l'Ordner sous GEM). Enfin, le desktop est désactivé et le SHELL S'occupe du chargement et de la mise en route de l'application. A la fin du déroulement du programme, le SHELL est de nouveau appelé et il active à nouveau le desktop ou bien il appelle une autre application.



Avant d'écrire un programme sur l'ATARI ST, il faut déterminer son apparence.
Ainsi, il faut faire la distinction entre une application et les accessory.

Nous appelons application un programme tout à fait normal comme par exemple un traitement de texte ou une gestion de fichier qui est chargé par le SHELL en mémoire centrale avant d'être exécuté.

Un accessory au contraire est une mini-application qui est chargée lors de l'activation du TOS dans le tampon des accessory et qui est mise en route de la même façon comme nous le verrons plus loin. Le rôle d'un accessory dans ce cas est d'attendre une instruction d'appel qui peut être exécutée à tout moment et plus particulièrement durant le déroulement d'une application.

Ensuite, nous devons décider dans quel environnement l'application doit travailler.

Il peut s'agir:
- du TOS
- du dessus de bureau
- d'une fenêtre définie auparavant.

Une application TOS est une application tout à fait normale. Le seul environnement est GEM. Les routines AES et VDI ne peuvent pas être utilisées.

La deuxième possibilité utilise l'AES et le VDI, mais la surface de travail ne pourra pas se limiter à une fenêtre. Elle couvrira tout l'écran. L'inconvénient est qu'un seul programme peut être exécuté et qu'on n'a pas accès à la fenêtre d'application ni au menu du desktop qui pourrait permettre par exemple l'appel d'un accessory.

Si le programme ouvre une fenêtre lors de son initialisation, il doit correspond au standard ATARI et le programmeur doit mettre en oeuvre de nombreuses routines supplémentaires.

Si on utilise complètement les fenêtres par exemple, il s'agira de l'accès au déplacement des fenêtres, à l'activation d'une fenêtre par cliquage, à l'agrandissement ou à la réduction d'une fenêtre par cliquage, etc...

4.1.4. LE RESOURCEFILE

Dans une application sous GEM, il faut encore s'occuper de relier le ressourcefile (.RSC) qui sert à adapter le logiciel à la langue maternelle de l'utilisateur.
Alors qu'il fallait à l'origine réecrire tout le programme et modifier les messages du code source pour ensuite le compiler de nouveau, il suffit maintenant de modifier le ressourcefile des logiciels sous GEM. Ce fichier binaire contient les message utilisateur ainsi que la structure de menu du programme (menu dialog et déroulants).

On peut simplement éditer ou créer le fichier RSC avec le RESSOURCE CONSTRUCTION SET dont nous parlerons plus tard. Naturellement, on peut aussi l'écrire à la main, si bien que les données correspondantes seront chargées dans un secteur mémoire particulier à la mise en route du programme et y seront modifiées et crées par le programme.

4.1.5 LE TRAVAIL SOUS TOS.

L'avantage de GEM est surtout de pouvoir adapter des programmes. Ainsi, on peut adapter sans problème sur l'ATARI ST les programmes en C développés sur un autre ordinateur. Mais un utilisateur qui n'a jamais écrit de programme en C devrait plutôt commencer par développer sous TOS où il n'aura pas à s'occuper d'appel du VDI et de l'AES.
C'est pour cette raison que nous allons vous présenter un listing simple en C.

4.2 LE VINGT ET UN

Dans ce jeu, l'objectif des deux joueurs, dont un est l'ordinateur, est d'atteindre le nombre vingt et un par l'addition de uns et de deux. 'Le vingt et un' est un jeu de NIM et peut donc être résolu par des règles mathématiques simples. Naturellement, nous n'allons pas nous plonger dans le domaine des mathématiques, ce qui ne nous permettrait pas de trouver une stratégie gagnante simple. Réfléchissons un peu: il y à un état de départ et un état d'arrivé définis (0 et 21). Les coups légaux sont l'addition de un et de deux. Le gagnant est celui qui aura atteint 21 en jouant le dernier coup.

L'observation de ces règles nous permet de diviser l'objectif à atteindre en plusieurs sous-objectifs. En effet, il ne faut pas laisser à notre opposant le nombre 19, ce qui lui permettrait d'atteindre 21. L'objectif d'arrivé est donc 19. Le même raisonnement nous permet d'atteindre comme prochain objectif le nombre 16.

De cette manière, on peut considérer les nombres 1, 4, 7, 10, 13, 16 et 19 comme positions gagnantes pour nous et perdantes pour l'opposant. Notre stratégie de jeu consiste alors à jouer le bon coup (1 ou 2) pour atteindre un nombre gagnant.

Avant de commencer à programmer, il faut d'abord s'intéresser à la structure du programme. Il s'agira d'un programme linéaire dont les parties seront l'initialisation (init), l'affichage du jeu (affichage), le coup du joueur (joueur) et le coup de l'ordinateur (computer), le traitement et la fin.

Dans la première partie du programme, le compteur est mis à zéro et on demande au joueur de choisir qui va jouer le premier.
La partie joueurs est constituée d'une simple fonction de saisie à choix multiple (getchar, switch case). GETCHAR est une fonction standard de la bibliothèque C qui sert à saisir un seul caractère du clavier.

Cet appel est largement suffisant pour le jeu car il suffit de saisir un seul chiffre. Ce chiffre est alors additionné au compteur, ce qui est effectué dans notre exemple après l'opérateur d'incrément ++ (état++ correspond à l'ordre BASIC état = état+1).

Avant que le second joueur ne puisse jouer dans la boucle principale du programme (while (etat<but)), il faut tester si on n'a pas atteint l'objectif terminal, ce qui est établi par == dans le cas de 21 : if (etat==but) break;, sinon le programme risquerait de continuer sans s'arrêter. BREAK sert en C à quitter une boucle ou une instruction CASE, comme à l'intérieur de la fonction computer(). Cette forme de choix multiple permet en C d'exécuter une opération tant qu'une des condition n'est pas remplie. Si on n'arrête pas une instruction CASE par BREAK, C continuerait à l'exécuter.

Dans la routine COUPCOMPUTER(), l'ordinateur tente d'atteindre un nombre gagnant. Si cela est impossible alors que le joueur a atteint un nombre gagnant, peu importe que l'ordinateur ait joué le coup +1 ou +2.

Si vous êtes débutant en C, les explications qui suivent vous intéresseront : Les programmes en C sont constitués d'une suite de fonctions. Lors du traitement, un branchement est toujours effectué en premier vers l'instruction MAIN(). Il faut donc trouver l'instruction MAIN().

En C, les variables peuvent être définies aussi bien globales que locales. Dans l'exemple du programme 21, il s'agit de variables globales. Elles ont été définies au début du programme avant les fonctions.
On y trouvera également les instructions #include et #define.

Avec #include, on ordonne au préprocesseur, qui est une partie du compilateur, de mettre le fichier stdio.h dans le fichier 21. Dans le cas d'une entré/sortie standard de fichier, il s'agit de définitions spécifiques à la machine qui permettent au code source de

fonctionner sur différentes machines. De cette manière, vous pouvez écrire des parties de programme dont vous serez sûr qu'elles fonctionneront parfaitement à la compilation et pendant l'exécution, ce qui vous permettra de vous constituer une bibliothèque de programmes.

Avec #define, vous définissez une constante symbolique, comme par exemple YES qui a la valeur 1 (vrai) et NO, 0. à ce qui n'est pas une condition vraie pour le code exécutable). Le préprocesseur donnera la valeur correspondante à chaque constante du programme source, ce qui permet d'écrire un programme plus lisible. Afin de distinguer les constantes des variables, nous vous conseillons de les écrire en majuscules.

```
/* 21 - JW 16.08.1985 */


#include "stdio.h"

#define YES 1
#define NO 0

int but ,etat ,sp,jeu  ;


main()
{
        hallo();
start:init();
        if (sp == YES)
        {
                affichage();
                player();
        }

        while(etat <but )
                {
                affichage();
                computer();
                if (etat  == but )
                        break;
                affichage();
                player();
                }
        fin ();
        printf("encore un jeu     ?\n");
        jeu  =getchar();
        if (jeu   == 'o')
            {
            goto start;
            }
}
```

```
hallo()

{
        printf("********** V I N G   E T   U N  **************\n");
        printf("Le but du jeu est d'atteindre la somme de 21 \n");
        printf("en additionnant au choix 1 ou 2 a la valeur .\n");
}


init()
{
        but =21;
        etat =0;
        printf("\n\nVoulew-vous commencer?");
        jeu  =getchar();
            if (jeu   == 'j')
                  sp = YES;
            else sp = NO;
        printf("\n");
}


affichage()
{
        printf("Etat du jeu: %d\n",etat );
}


player()
{
    sp = YES;
         jeu  =0;
         printf("\nVoulez-vous augmenter de 1 ou de 2 ?\n");
         jeu   = (getchar() - '0');
         switch(jeu  )
         {
             case 1 :
             {
```

```
                printf("\nOkay !\n");

                etat ++;
                break;
            }
            case 2 :
            {
                printf("\nBon aussi!\n");
                etat ++;
                etat ++;
                break;
            }
            default :
            {
                printf("\nPas autant, voyons    !!\n");
                player();
            }
        }
}


computer()
{
    sp = NO;
        switch(etat )
        {
            case 2:
            case 5:
            case 8:
            case 11:
            case 14:
            case 17:
            case 20:
            {
                plusun  ();
                break;
            }
            case 1:
```

```
            case 4:

            case 7:
            case 10:
            case 13:
            case 16:
            case 19:
            {
                printf("\nJ'augmente de  2.\n");
                etat ++;
                etat ++;
                break;
            }
            default:
            {
                plusun  ();
            }
        }
}


plusun  ()
{
        printf("\nJ'augmente de  1.\n");
        etat ++;
}


fin ();
{
    if (sp == YES)
        printf("\n\nBonne chance           .\n\n");
    else
        printf("\n\nJ'ai gagné cette fois çi... \n\n");
}
```

4.3. LE PAS SUIVANT : UNE APPLICATION GEM

Maintenant que vous connaissez le langage C et que des concepts comme les fichiers includes et les constantes symboliques n'ont plus de secret pour vous, nous allons réaliser pas à pas une parfaite application sous GEM.
Comme nous l'avons indiqué dans les pages précédentes, GEM et en particulier le Virtual Device Interface représentent une interface normée et relativement facile à manipuler pour les différents périphériques graphiques. Dans ce cas, peu importe que le programme doive contrôler un écran, ou une imprimante matricielle ou un traceur. En effet, les codes de contrôle spécifiques sont pris en charge par le VDI. Le programmeur n'a donc pas à s'occuper de ce travail. Naturellement, le VDI doit connaître les données spécifiques aux périphériques, comme le pas minimum dans les directions x/y, le nombre de stylos, etc...
A cet effet, on peut charger différents drivers de périphériques. Pour l'instant, GEM ne connaît que le driver d'écran pour l'ATARI ST.

Afin d'indiquer au VDI ce que l'on attend de lui, on utilise une routine qui nécessite cinq paramètres, qui sont en fait cinq vecteurs de données :

- le vecteur de contrôle (contrl)
- le vecteur de saisie (intin)
- le vecteur de saisie des coordonnées de points (ptsin)
- le vecteur de sortie (intout)
- le vecteur de sortie pour les coordonnées des points(ptsout)

Il faut fixer les paramètres nécessaires pour la fonction à exécuter avant d'appeler le VDI.

Les programmeurs en langage machine devront charger ces valeurs à la main, alors que les utilisateurs de langages évolués disposeront de bibliothèques.
Ceci simplifiera grandement le travail et permettra de le normer, comme nous le verrons plus tard.

Tous les vecteurs font deux octets, c'est pourquoi nous définirons toutes les variables correspondantes en C en tant qu'integer. Dans les programmes de démonstration qui suivent, vous trouverez les définitions au début de la déclaration des variables globales.

Le premier pas d'un programme sous GEM consiste, après l'initialisation de ces vecteurs de contrôle, à fixer les paramètres pour la station de travail utilisée et d'ouvrir celle ci pour l'utilisation à venir.
Pour cela est prévue la routine VDI OPENWORKSTATION.
Son appel sert à charger le driver servant à mettre le périphérique de sortie en mode graphique (ce n'est pas encore possible sur le ST) et à l'initialiser. Nous pouvons donc établir lors de l'appel openworkstation que les lignes doivent être dessinées en discontinu et que la couleur à utiliser est le noir. Cela signifie pour les programmeurs qu'il faudra écrire le paramètre dans le vecteur contrl. Il s'agit :

1. du type de la ligne (discontinue, oblique...)
2. couleur de la ligne
3. type du marqueur
4. couleur du polymarqueur
5. type d'écriture
6. couleur de l'écriture
7. motif pour les dessins de polygônes
8. motif de remplissage
9. couleur de remplissage

Heureusement, la valeur normale de ces paramètres est 1,ce qui permet de faire l'initialisation avec une boucle.

L'exception est la valeur 10 dans int_in. Avec ce paramètre on fixe le système de coordonnées. Vous le savez déjà, on peut utiliser soit les coordonnées existantes soit des coordonnées normalisées. Etant donné que nous avons besoin de rapidité et que nous écrivons notre programme pour le moniteur, nous donnons la valeur 2 pour les coordonnées raster (un nul choisira le NDC, le un est réservé pour une application ultérieure).

```
open_vwork()
{
int i;
      for (i = 1; i <10; i++){
        int_in[i] = 1;
        }
      int_in[10] = 2;
      v_opnvwk(int_in, &handle, int_out);
}
```

L'appel v_opnvwk(int_in, &handle, int_out) exécute l'initialisation. Le résultat est constitué de nombreux paramètres dans le vecteur int_out, qui ne nous intéresse pas à ce moment. Nous ne nous intéressons qu'à la valeur donnée à handle. En effet, grâce à ce paramètre, nous pouvons correspondre parfaitement au secteur de travail de cette application.

Si vous regardez main() dans le programme qui suit, vous remarquerez que deux appel CALL y sont exécutés.

Appl_init prépare un vecteur de contrôle pour l'utilisation de l'AES et on obtient comme résultat un code d'identification de l'application qui fait l'appel. Ce code est important avant tout lorsque plusieurs programmes se partagent la mémoire. Ils peuvent être mis en correspondance à l'aide de ap_id. Graf_handle enfin lit le paramètre du desktop et peut préparer cette valeur pour une utilisation future. Etant donné que tous ces paramètres ne nous sont d'aucune utilité, nous les mettons dans une variable quelconque. Draw() enfin est notre programme principal. Il trace le contour d'une maison.

Tout programme doit avoir une fin correcte. L'application doit libérer la place mémoire qu'elle a utilisé, rendre la main au poste de travail pour réactiver le desktop.

REMARQUE sur le listing qui suit : excepté draw(), vous devriez mettre les parties du programme dans un fichier et le définir en tant que startfile. Si vous voulez plus tard écrire vos propres programmes, dans lesquels vous pourrez vous passer des fenêtres (par exemple démonstrations graphiques), vous pourrez utiliser les codes sources sans modification et écrire vos propres routines à partir de draw(). Pour terminer correctement votre application, votre dernière ligne de programme devra être :

desktop();

D'autre part, le listing a été étendu à la fonction click() qui attend que la touche gauche de la souris soit pressée. Sinon, vous n'auriez pas le temps de voir l'image.

```
/*****************************************************************************/
/**************              programm: maison1.c                 *************/
/************ dessine une maison et attend clic gauche de la souris ********/
/**************              JW octobre 1985                     *************/
/*****************************************************************************/
                                                  /* include files          */
#include "obdefs.h"                 /* on les prends tous car qui sait      */
#include "define.h"                 /* ce qui sera utilisé                  */
#include "gemdefs.h"
#include "osbind.h"
#include "gembind.h"

                                                  /*variables globales       */
int contrl[12];
int intin[128];
int ptsin[128];
int intout[128];
int ptsout[128];                    /* suffisament de place pour tous les cas*/

int handle,i;                       /* virtual workstation handle           */
int phys_handle;                    /* physical workstation handle          */
int pxyarray[12];                   /* vecteur pour coordonnées x,y         */
int int_in[11];                     /* saisie dans vecteur gsx              */
int int_out[57];                    /* sortie du vecteur GSX                */
```

```
int ap_id;                          /* reconnaissance de l'application      */

int dummy;

main()
{
        ap_id=appl_init();          /* initialise    GEMAES Array-Structures */
        handle=graf_handle(&dummy,&dummy,&dummy,&dummy);
                                                /* manipulation du Desktops    */
        open_vwork();               /* mise en place du secteur de travail   */
        graf_mouse(256,&dummy);     /* cache la souris                       */
        draw();                     /* dessin de l'oeuvre                    */
        v_gtext(handle,1,350,"pressez à gauche SVP...");
        click();                    /* attente du clic du bouton gauche      */
        desktop();                  /* fin du programme                      */
}


open_vwork()
{
int i;
      for (i = 1; i <10; i++){
           int_in[i] = 1;            /* init int_in array: linetype, couleur, */
           }                         /* fillstyles etc.                       */
        int_in[10] = 2;             /* utilise coord.RC                      */
        v_opnvwk(int_in, &handle, int_out); /* on peut y aller      ...      */
}


desktop()
{
        v_clsvwk(handle);           /* quitter workstation                   */
        appl_exit();                /* plus    d'appels GEM                  */
}
```

```
click()                                 /* attend cliquage de la souris (gauche) */

{
        evnt_button(1,1,1,&dummy,&dummy,&dummy,&dummy);
}



/*---------------- les parties de programme se trouvent à partir d'ici------*/


draw()
      {
        int style;                      /* Variable pour motif de remplissage   */
        style = 3;                      /* choix du motif                       */
        pxyarray[0] = 100;              /* coordonnée x du point 1              */
        pxyarray[1] = 100;              /* coordonnée y du point 1              */
        pxyarray[2] = 100;              /* point 2                              */
        pxyarray[3] = 300;
        pxyarray[4] = 500;              /* point 3                              */
        pxyarray[5] = 300;
        pxyarray[6] = 500;
        pxyarray[7] = 100;
        pxyarray[8] = 300;
        pxyarray[9] = 50;
        pxyarray[10] = 100;
        pxyarray[11] = 100;

        v_pline(handle, 6, pxyarray); /* poligone dans secteur de travail     */
                                      /* 6 points avec coord dans pxyarray    */

        vsf_interior(handle, style);  /* set fill interior style: solid/hollow*/
        v_fillarea(handle, 6, pxyarray); /* remplir surface recouverte par    */
                                         /* polygone                           */
}
```

Avec cela nous pouvons exécuter une application sur le desktop. Le prochain pas que nous devons prendre en charge est de faire une routine open_window() qui nous permette d'utiliser une fenêtre en tant que secteur de travail.

Réfléchissons à la taille que devra avoir notre fenêtre. Avec GEM, on convient que le coin gauche en haut est l'origine d'un objet et que la hauteur et la largeur sont indiqués en points.
Mais qui désirerait compter des points ? Enfin, le desktop est une fenêtre dont la taille est maximale. Pourquoi ne pas la mesurer alors que le VDI offre une fonction correspondante avec wind_get.
Tout ce que nous devons savoir est le numéro de la fenêtre du desktop. Or elle est égale à zéro dans tous les cas.

Remplaçons main() du programme MAISON1 par l'appel

```
wind_get(0,WF_WORKXYWH,&xdesk,&ydesk,&wdesk,&hdesk);
```

Il ne faudra pas oublier évidemment de mettre les nouvelles variables dans la liste d'initialisation :

```
int xdesk,ydesk,wdesk,hdesk;
```

Lisons la fonction wind_create() qui donne le numéro de la fenêtre (wi_handle). Les paramètres de la fonction wind_create sont d'une part les coordonnées conventionnelles et d'autre part, les éléments de contrôle pour notre fenêtre. Ainsi chaque partie de la fenêtre et chaque particularité sont représentées par un bit positionné dans un nombre binaire :

0x0001	NAME :	ligne titre avec son nom
0x0002	CLOSER :	champ final
0x0004	FULLER :	champ pour la taille maximale (en haut à gauche)
0x0008	MOVER :	la fenêtre peut être déplacée
0x0010	INFO :	ligne d'information (p.ex. : 123456 bytes used in)

0x0020 SIZER : Champ d'agrandissement (en bas à gauche)
0x0040 UPARROW :flèche vers le haut
0x0080 DNARROW flèche vers le bas
0x0100 VSLIDE : déplacement vertical
0x0200 LFARROW :flèche vers la gauche
0x0400 RTARROW :flèche vers la droite
0x0800 HSLIDE : déplacement horizontal.

Si votre fenêtre doit contenir une ligne titre à cause de la limitation de la surface de travail, votre premier paramètre lors de l'appel de wind_create doit être 1. Si vous voulez une ligne de fin pour terminer convenablement votre application, il faut programmer un 3 (1+2; bits positionnés : 00000011).

Il est maintenant inutile d'être obsédé par les bits. On peut aussi utiliser des constantes symboliques qui seront définies dans un fichier #include.
Les symboles décrits plus haut sont courants car ils se trouvent dans le fichier GEMBIND.H du C. Le programmeur peut donc utiliser simplement une constante symbolique et lui assigner la valeur désirée :

define WI_KIND (SIZER MOVER FULLER CLOSER NAME)

Ainsi on peut établir le format de la fenêtre sans problème. On fixe ensuite le nom, ce que permet entre autres la fonction wind_set() et enfin, la fenêtre peut être ouverte.

Nous avons réalisé ce processus dans une fonction qui peut être reliée sans problème à un programme, (et qui grâce aux constantes symboliques reste la même pour toute fenêtre):

```
open_window()
{
      wi_handle=wind_create(WI_KIND,xdesk,ydesk,wdesk,hdesk);
      wind_set(wi_handle, WF_NAME, 'Trucs et Astuces' ,0,0);
      wind_open(wi_handle,xdesk,ydesk,wdesk,hdesk);
}
```

Nous avons ainsi ouvert notre fenêtre sur l'écran mais beaucoup d'éléments ne sont pas utilisables. Le test de la manipulation que l'utilisateur désire effectuer sur la fenêtre est très simple à mettre en oeuvre. On dispose en effet pour cela d'une partie de l'AES qui est l'EVENT-LIBRARY.

Elle offre de nombreuses routines de test permettant d'obtenir certains résultats. Il peut s'agir d'un test de la souris, tel le event_button dont vous avez fait connaissance dans click(). Mais ce peut être également un résultat qui peut apparaître à une fenêtre par un message.

La structure d'un tel message doit évidemment être établie exactement, ce qui peut être réalisé par le message protocol de l'AES sous GEM. On utilise pour cela un autre vecteur, le tampon de message (msgbuff) qui a une longueur maximum de 8 données. Le protocol vérifie que le message se trouve dans msgbuff(0). msgbuff(1) contient le code d'identification de l'application et les autres données donnent des informations sur des paramètres importants.

Si l'utilisateur choisi par exemple un point particulier du menu avec la souris, l'AES inscrit dans msgbuff(0) un 10, qui est le code de MN_SELECTED. msgbuff(3) donne le numéro du menu concerné (par exemple DESK, FILE) et msgbuff(4) indique le numéro de l'objet concerné (CONTROL, FORMAT). Lorsque les numéros sont fixés, on peut exécuter rapidement l'action désirée à l'aide d'une structure CASE.

Etant donné qu'il est écrit en majuscules, vous considérez peut être MN_SELECTED comme la description d'une constante qui est définie dans un quelconque fichier include. En fait, dans ce cas encore, on ne s'ennuie pas avec des numéros qu'il faudra bientôt changer, mais on peut fixer des noms faciles à interpréter pour une édition ultérieure :

MN_SELECTED :	choix du point du menu
WM_REDRAW :	la fenêtre doit être redessinée
WM_TOPPED :	cette fenêtre doit être activée
WM_CLOSED :	la ligne de fin a été mise en place
WM_FULLED :	mise en place de la plus grande taille
WM_ARROWED :	une flèche a été cliquée
WM_HSLID :	mise en place du déplacement horizontal
WM_VSLID :	mise en place du déplacement vertical
WM_SIZED :	la taille de la fenêtre a été modifiée
WM_MOVED :	la fenêtre a été déplacée
WM_NEWTOP :	la fenêtre a été activée
AC_OPEN :	est envoyé vers l'accessory choisi dans le menu du desktop
AC_CLOSE :	est envoyé vers l'accessory à fermer

Comme vous le constatez, la liste est longue. Et dès que vous désirerez manipuler certaines particularités des fenêtres GEM, vous devrez écrire des routines pour gérer tous ces résultats.

Heureusement, cela peut presque toujours être réalisé à l'intérieur d'un bloc de fonctions (avec une grande structure de choix conditionnel). Vous devrez simplement faire attention à ce que votre application soit effectivement en attente de tels résultats sans arrêt.

A cet effet, vous utiliserez un des event_multi() calls avec lequel vous pourrez attendre un message, une action de la souris ou du clavier. Nous précisons encore une fois que le déroulement est toujours réalisé de cette manière, c'est à dire que vous devez toujours indiquer à GEM ce que vous attendez. Oubliez le processus selon lequel le programme doit attendre une action (Ce à quoi vous êtes peut être habitué par des application TOS courantes comme le 21).

Pensez à ce truc chaque fois que vous écrivez un programme qui semble être bloqué.

Car tant que vous pouvez déplacer la souris (ce que vous ne pouvez contrôler naturellement que si vous ne l'avez pas rendue invisible), l'ATARI est occupé à une tache quelconque et n'est pas planté. Vous avez simplement oublié de lui indiquer qu'il faut attendre des réactions extérieures. Le seul moyen que vous ayez pour le stopper est la touche RESET.
Pour cela, votre programme doit fonctionner dans une grande boucle qui n'est quitté que si une condition est remplie. Ce peut être le cliquage de la souris ou bien une commande de la ligne de fin.
Dans ce cas, le programme devra attendre un message du type WM_ CLOSED:

```
do(
        event_multi(...);
        controledefenetre;
        votre programme est inscrit ici;
        )while ligne de fin n'a pas été perturbée
```

Ce système est typique d'une application. Un accessory est bâti selon une structure différente. Mais nous en verrons plus par la suite. Nous voulons plutôt donner un cadre à notre programme de démonstration vous permettant d'agrandir, de rapetisser et de déplacer la fenêtre à volonté.

```
/*****************************************************************************/
/**************          Programme: MAISON3.C                  *************/
/**************      Manipulation de fenetres                  *************/
/**************            JW Octobre 1985                     *************/
/*****************************************************************************/

                                    /* include files                        */
#include "obdefs.h"                 /* D'abord tous- On ne sait jamais      */
#include "define.h"                 /* ce aui sera utilisé                  */
#include "gemdefs.h"
#include "osbind.h"
#include "gembind.h"

                                    /* Definition pour utilisation future    */

#define WI_KIND   (SIZER|MOVER|FULLER|CLOSER|NAME)
                                    /* fenetre de travail, titre, champde fin*/
#define MIN_WIDTH  (2*gl_wbox)
#define MIN_HEIGHT (2*gl_hbox)

extern int gl_apid;
                                    /*variables globales                     */
int contrl[12];
int intin[128];
int ptsin[128];
int intout[128];
int ptsout[128];                    /* suffisamment de place pour tous les cas*/

int handle,i;                       /* virtual workstation handle            */
int phys_handle;                    /* physical workstation handle           */
int pxyarray[12];                   /* Vecteur pour coord. X et Y            */
int int_in[11];                     /* Chargement dans vecteur int_in       */
int int_out[57];                    /* Affichage du vecteur GSX              */

int wi_handle;                      /* Manipulation de la fenetre d'application*/
int top_window;                     /* fenetre tout en haut                  */
int xdesk, ydesk, wdesk, hdesk;     /* Parametre de la taille de la fenetre  */
int xold, yold, hold, wold;
int xwork, ywork, hwork, wwork;
```

```
int mx, my;                          /* coord. x et y de la souris          */
int butdown;

int ap_id;                           /* reconnaissance de l'application     */
int menu_id;                         /* reconnaissance de la fenetre de travail*/
int fulled;
int hidden;

int msgbuff[8];                      /* event message buffer                */
int keycode;                         /* contient code char d apres evnt_keybrd */

int gl_wchar, gl_hchar;              /* Hauteur du caractere                */
int gl_wbox, gl_hbox;

int dummy;



/*****************************************************************/
/* initialisation utile                                          */
/*****************************************************************/
open_vwork()
{
int i;
      for (i = 1; i <10; i++){
        int_in[i] = 1;               /* init int_in array: linetype, farbe,  */
        }                            /* fillstyles etc.                      */
      int_in[10] = 2;                /* utilise coord. RC                    */
      handle=phys_handle;
      v_opnvwk(int_in, &handle, int_out); /* On peut y aller      ...        */
}


/*****************************************************************/
/* open window                                                   */
/*****************************************************************/
open_window()
{
```

```
    wi_handle=wind_create(WI_KIND,xdesk,ydesk,wdesk,hdesk);
    wind_set(wi_handle, WF_NAME," la maison T&T    ",0,0);
    graf_growbox(xdesk+wdesk/2,ydesk+hdesk/2,gl_wbox,gl_hbox,xdesk,ydesk,wdesk,hdesk);
    wind_open(wi_handle,xdesk,ydesk,wdesk,hdesk);
    wind_get(wi_handle,WF_WORKXYWH,&xwork,&ywork,&wwork,&hwork);
}


/*****************************************************************/
/* afficher/cacher souris                                        */
/*****************************************************************/
show_mouse()
{
      graf_mouse(257,&dummy);
}

hide_mouse()
{
      graf_mouse(256,&dummy);
}


/*****************************************************************/
/* mise en place de clipping-Parametre                           */
/*****************************************************************/
set_clip(x,y,w,h)
int x,y,w,h;
{
int clip[4];
   clip[0]=x;
   clip[1]=y;
   clip[2]=x+w;
   clip[3]=y+h;
   vs_clip(handle,1,clip);
}


/*****************************************************************/
/* redessiner apres manipulation de fenetre                      */
/*****************************************************************/
```

```
do_redraw(xc,yc,wc,hc)
int xc,yc,wc,hc;
{
GRECT t1,t2;

   hide_mouse();
   wind_update(TRUE);
   t2.g_x=xc;
   t2.g_y=yc;
   t2.g_w=wc;
   t2.g_h=hc;
   wind_get(wi_handle,WF_FIRSTXYWH,&t1.g_x,&t1.g_y,&t1.g_w,&t1.g_h);
   while (t1.g_w && t1.g_h)
      {
      if (rc_intersect(&t2,&t1))
         {
         set_clip(t1.g_x, t1.g_y, t1.g_w, t1.g_h);
         draw_haus();
         }
      wind_get(wi_handle,WF_NEXTXYWH,&t1.g_x,&t1.g_y,&t1.g_w,&t1.g_h);
      }
     wind_update(FALSE);
     show_mouse();
}

/*********************************************************************/
/*  test de resultats :      Window, souris , Keyboard               */
/*********************************************************************/
multi()
{
int event;

      do{
      event = evnt_multi(MU_MESAG | MU_BUTTON | MU_KEYBD,
         1,1,butdown,
         0,0,0,0,0,
```

```
/*****************************************************************/
/* WINDOW() : manip fenetre    : deplacement, agrandissement, etc*/
/*****************************************************************/

  wind_update(TRUE);

  if (event & MU_MESAG)
    switch (msgbuff[0]) {

    case WM_REDRAW:
      do_redraw(msgbuff[4],msgbuff[5],msgbuff[6],msgbuff[7]);
      break;

    case WM_NEWTOP:
    case WM_TOPPED:
      wind_set(wi_handle,WF_TOP,0,0,0,0);
      break;

    case WM_SIZED:
    case WM_MOVED:
      if(msgbuff[6]<MIN_WIDTH)msgbuff[6]=MIN_WIDTH;
      if(msgbuff[7]<MIN_HEIGHT)msgbuff[7]=MIN_HEIGHT;
      wind_set(wi_handle,WF_CURRXYWH,msgbuff[4],msgbuff[5],msgbuff[6],msgbuff[7]);
      wind_get(wi_handle,WF_WORKXYWH,&xwork,&ywork,&wwork,&hwork);
      break;

    case WM_FULLED:
      if(fulled){
      wind_calc(WC_WORK,WI_KIND,xold,yold,wold,hold,
          &xwork,&ywork,&wwork,&hwork);
      wind_set(wi_handle,WF_CURRXYWH,xold,yold,wold,hold);}
        else{
      wind_calc(WC_BORDER,WI_KIND,xwork,ywork,wwork,hwork,
          &xold,&yold,&wold,&hold);
      wind_calc(WC_WORK,WI_KIND,xdesk,ydesk,wdesk,hdesk,
          &xwork,&ywork,&wwork,&hwork);
      wind_set(wi_handle,WF_CURRXYWH,xdesk,ydesk,wdesk,hdesk);
        }
      fulled ^= TRUE;
      break;
```

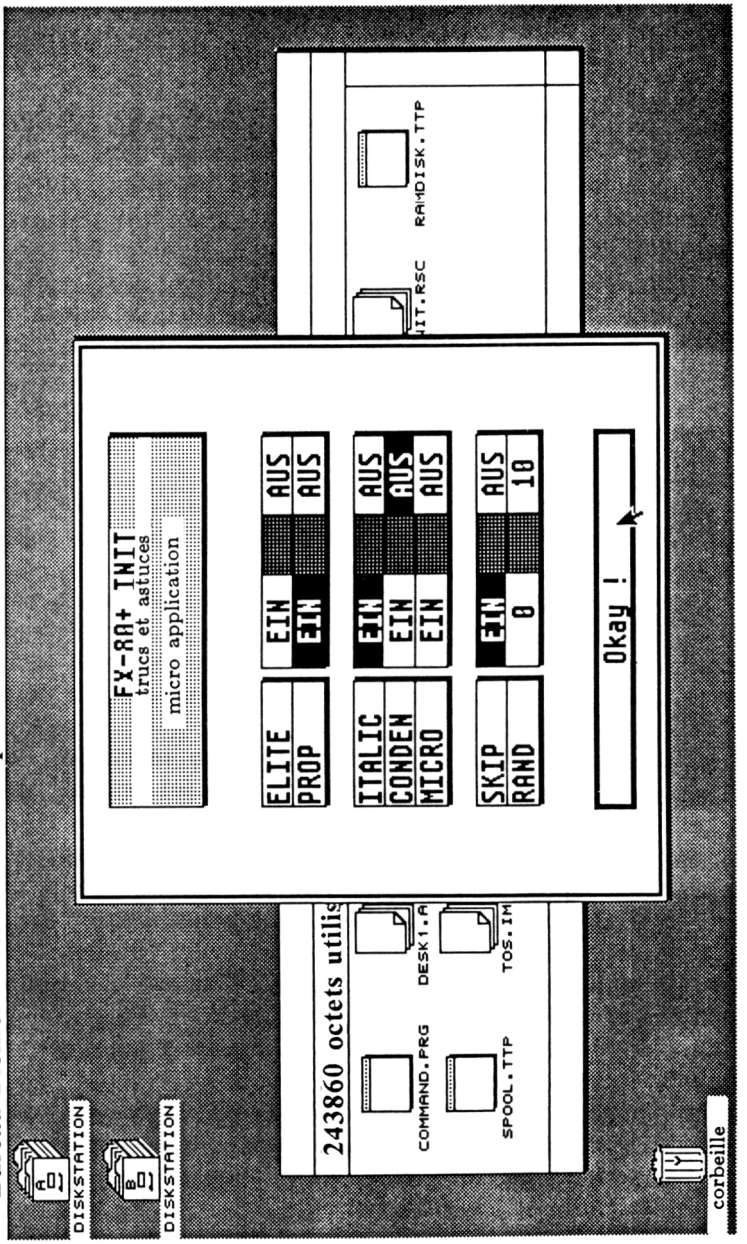
```
    } /* switch (msgbuff[0]) */

  if ((event & MU_BUTTON)&&(wi_handle == top_window))
    if(butdown) butdown = FALSE;
    else butdown = TRUE;

    if(event & MU_KEYBD){
       do_redraw(xwork,ywork,wwork,hwork);
    }

  wind_update(FALSE);

    }while(!((event & MU_MESAG) && (msgbuff[0] == WM_CLOSED)));
                              /* champ de fin touche                    */
     wind_close(wi_handle);
     graf_shrinkbox(xwork+wwork/2,ywork+hwork/2,gl_wbox,gl_hbox,xwork,ywork,wwork,hwork);
     wind_delete(wi_handle);      /* liberer mémoire                    */
     v_clsvwk(handle);            /* rendre la main à station de travail */
     appl_exit();                 /* et à desktop                       */



main()
{
     ap_id=appl_init();           /* initialise GEMAES Array-Structures  */
     phys_handle=graf_handle(&gl_wchar, &gl_hchar, &gl_wbox, &gl_hbox);
                                  /* manipulation du Desktop             */
     wind_get(0,WF_WORKXYWH, &xdesk, &ydesk, &wdesk, &hdesk);
     open_vwork();                /* mise en place de secteur de travail */
     open_window();               /* ouvrir fenetre d'application        */
     graf_mouse(ARROW,&dummy);    /* forme de souris                     */
     v_gtext(handle,10,gl_hchar*3,"c'est la maison t&t .");

     hidden=FALSE;
     fulled=FALSE;
     butdown=TRUE;
     multi();                      /* que fait l'utilisateur ?           */

}
```

```
/*----------------- A partir d'ici se trouve la parrtie de programme---------*/


draw_haus()
      {
      int style;                    /* Variable pour motif de remplissage   */
      style = 3;                    /* choix du motif de remplissage        */
      pxyarray[0] = 100;            /* x-coord. point 1                     */
      pxyarray[1] = 100;            /* y-coord. point 1                     */
      pxyarray[2] = 100;            /* point 2                              */
      pxyarray[3] = 300;
      pxyarray[4] = 500;            /* point 3                              */
      pxyarray[5] = 300;
      pxyarray[6] = 500;
      pxyarray[7] = 100;
      pxyarray[8] = 300;
      pxyarray[9] = 50;
      pxyarray[10] = 100;
      pxyarray[11] = 100;

      v_pline(handle, 6, pxyarray); /* dessin polygone dans secteur travail */
                                    /* 6 points avec coord. dans  pxyarray  */

      vsf_interior(handle, style);  /* set fill interior style: solid/hollow*/
      v_fillarea(handle, 6, pxyarray); /* surface fermée du polygone        */
}
```

4.3.1. PRINIT : UNE APPLICATION

Maintenant que le travail de préparation est terminé, nous allons nous intéresser à une application utile, en gardant à l'esprit qu'elle pourra être installée ultérieurement à l'intérieur du DESK sur le menu.
Il s'agit de PRINIT, un court programme de mise en place de l'imprimante. Il est sans aucun doute très utile en tant qu'accessory car on doit souvent changer la police de caractères de l'imprimante et on est obligé de charger le BASIC pour envoyer une suite de CHR$. Avec PRINIT, il suffit de cliquer sur ELITE, puis sur RAND10 et de nouveau sur OKAY pour obtenir une impression de qualité.

Un tel mode de choix est très pratique mais il faut tout d'abord créer une 'dialogbox'. Ce qui nous intéresse est de savoir sur quel objet se trouve la souris dans cette boite.
Pour répondre à cette question il faut d'abord résoudre le premier problème. GEM gère toutes ces ressources sous la forme structurée d'un arbre. Cela veut dire que chaque objet possède une racine et que de ce point partent de nombreuses possibilités jusqu'à ce qu'un menu soit rempli d'objets.
Imaginez un rectangle qui contient deux petits rectangles l'un en dessous de l'autre. Le rectangle inférieur contient encore un texte quelconque.
La racine de cet arbre est alors le plus grand rectangle. Elle apparaît clairement et peut être décrite par des données typiques. Il s'agit entre autres des coordonnées du coin gauche en haut, de la largeur et de la hauteur.
De plus, la boite peut être décrite par l'épaisseur de son cadre ainsi que par la couleur de fond ou des signes.
Considérez simplement pour l'instant qu'une telle boite s'appellera G BOX.
Si nous considérons cette boite comme PARENT, ce qui est vrai pour une construction de ce type, CHILD sera une des boîtes à l'intérieur. La boite supérieure est vide, donc il s'agit d'après ce que nous savons d'une G BOX.


Bureau Fichier Visualisation Options
DISKSTATION
DISKSTATION
corbeille
243860 octets utilis
COMMAND.PRG
SPOOL.TTP
DESK1.A
TOS.IM
FX-80+ INIT
trucs et astuces
micro application
ELITE
PROP
EIN
EIN
AUS
AUS
ITALIC
CONDEN
MICRO
EIN
EIN
EIN
AUS
AUS
AUS
SKIP
RAND
EIN
0
AUS
10
Okay !
IIT.RSC
RAMDISK.TTP

La boite inférieure est une boite avec du texte. Elle est donc décrite par le type :

G_BOXTEXT.

Dès que l'on est fixé sur la structure de son menu et que l'on a établi une liste des données nécessaires pour la description d'un objet, on peut introduire ces données dans le programme et commencer à construire l'arbre à l'aide de OBJC_ADD.

Enfin, un OBJC_DRAW de l'arbre correspondant apparaît sur le périphérique de sortie. Comptez combien d'objets contient l'arbre 'dialog' construit pour PRINIT : il y en a 34 exactement. Combien de temps cela prendrait-il pour établir les données exactes?

Le Ressource Construction Set offre une aide rapide. Avec ce programme d'aide, on peut construire des structures de menu très rapidement et les éditer éventuellement plus tard. En effet le RCS crée des RCS.files qui contiennent toutes les données nécessaires. Ils sont chargés durant le fonctionnement du programme par la fonction rsrc_load(filename) dans une place mémoire réservée.
Dans ce qui suit, nous allons vous expliquer pas à pas comment construire le fichier PRINIT.RSC.

4.4. CONSTRUCTION D'UN FICHIER RSC

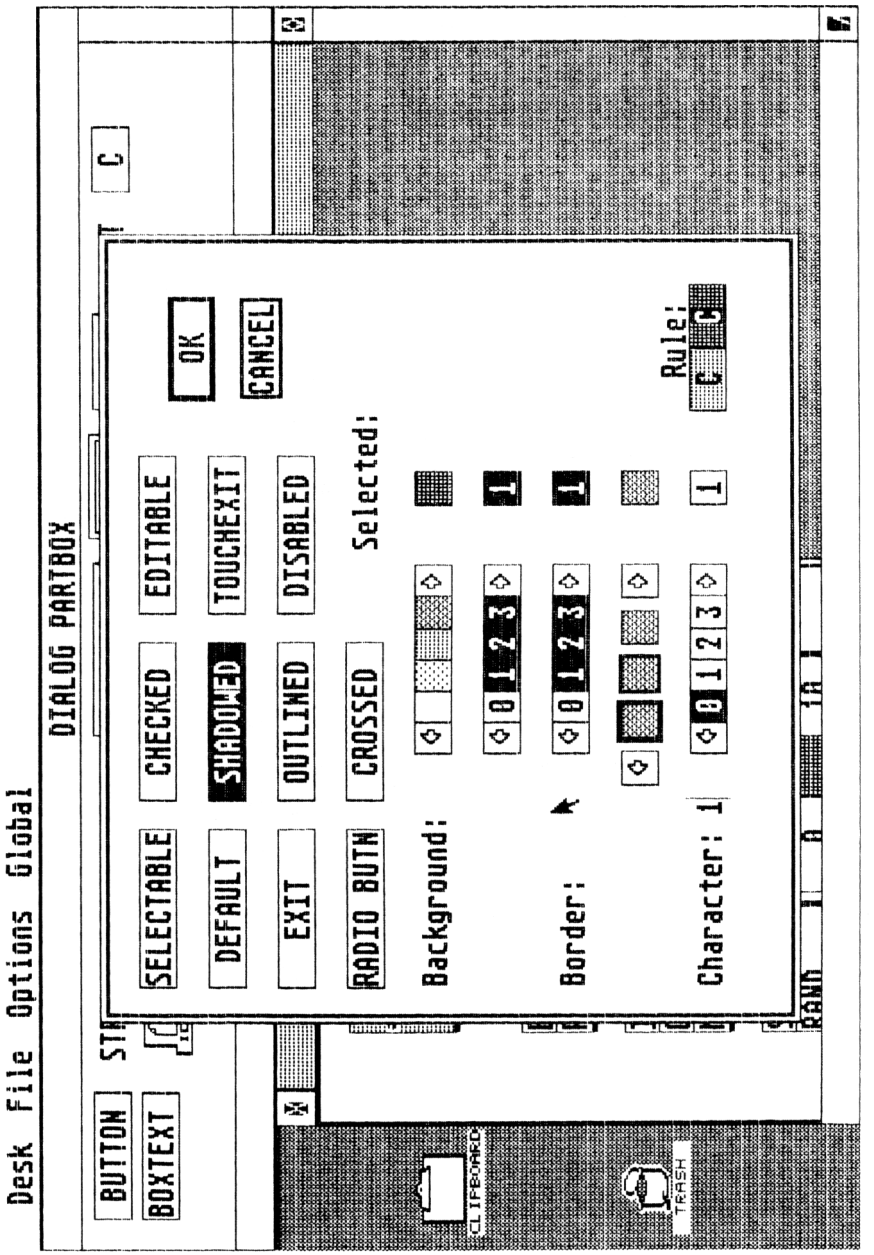
Une fois que le RCS a été chargé, vous voyez deux fenêtres à l'écran. Dans la fenêtre supérieure, la RESOURCE PARTBOX, se trouvent les composants disponibles. Après l'initialisation, il peut s'agir bien sûr des multiples structures en arbre.
Vous devez alors choisir si vous voulez construire un menu qui se trouvera plus tard dans la liste de menu du DESKTOP ou bien si vous voulez construire un arbre de choix dans lequel l'utilisateur pourra choisir entre plusieurs possibilités.
Ces deux types sont les plus utilisés, mais il existe également le ALERT-TREE qui ressemble à l'arbre de choix mais qui n'offre pas autant d'options.

D'autre part, le RCS dispose de l'arbre FREE qui ne présente pratiquement aucune restriction. La seule condition qui préside à la construction de cet arbre est qu'aucun objet ne peut en dépasser un autre alors que les autres exigent des règles de format spécifiques.
L'arbre symbolisé par un point d'interrogation sert simplement à réserver de la place jusqu'à ce que le programmeur sache ce qu'il fait et lui donne un nom.
S'il reste quelque part dans le resourcefile un autre arbre de type inconnu, vous pouvez être sûr que le programme se plantera.
Nous allons suite à ces explications cliquer sur le symbole pour un arbre dialog et en faire une copie dans notre fenêtre, c'est à dire le tirer dans la fenêtre en maintenant le bouton de la souris pressé. Le RCS affiche immédiatement une dialogbox et nous demande de nommer l'arbre. Entrez FXMENU et saisissez en pressant la touche RETURN ou bien en cliquant sur la box OKAY.

Enfin, vous déplacez le pointeur de souris sur la fenêtre inférieure et appuyez sur le bouton gauche de la souris pour l'activer. Cliquez deux fois sur l'arbre dialog décrit par FXMENU.
Choisissez dans la fenêtre supérieure le composant BOXTEXT et tirez en une copie dans la fenêtre inférieure FXMENU. BOXTEXT est une simple boite qui contient un texte et que nous utilisons pour nommer les différentes options (ELITE, ITALIC...). Nous aurons besoin d'au moins sept boîtes de ce type. Mais il est inutile d'activer à chaque fois une des deux fenêtre, alors que nous pouvons utiliser un moyen de copie du RCS. Pour cela, vous devez cliquer sur la boite contenue dans la fenêtre inférieure et la déplacer à la position désirée, mais en maintenant une des touches SHIFT pressée. Reprenez ce processus jusqu'à ce que vous ayez atteint le nombre de BOXTEXT désiré.

Chaque interrupteur on/off du FXMENU est constitué d'une grande boite (du le troisième élément à partir de la droite dans la partbox). Amenez en une dans la fenêtre inférieure et agrandissez la au format désiré. cliquez deux fois sur cette boite ou bien choisissez open dans le file-menu.

Desk File Options Global
RESOURCE PARTBOX
FREE
MENU
DIALOG
You may change the type and name of this tree. CAUTION: Choosing an incorrect type may have unexpected effects when editing. If in doubt, select DIALOG.
UNKNOWN
NAME: FXMENU
FREE
MENU
DIALOG
ALERT
OK
CANCEL
DIALOG
TREE1
CLIPBOARD
TRASH


Desk File Options Global
DIALOG PARTBOX
BUTTON
BOXTEXT
SELECTABLE
DEFAULT
EXIT
RADIO BUTN
CHECKED
SHADOWED
OUTLINED
CROSSED
EDITABLE
TOUCHEXIT
DISABLED
OK
CANCEL
Selected:
Background:
Border:
Character: 1
Rule:
C
CLIPBOARD
TRASH

Le RSC vous propose une autre dialogbox à l'intérieur de laquelle vous fixez les paramètres pour l'apparence de la boite. Il faut suivre la démarche suivante.

Chaque ON et OFF est mis en place par un BUTTON. Choisissez un BUTTON dans la partbox et positionnez le sur un coté à l'intérieur de la boite crée précédemment. Enfin, déplacez en une copie de la taille correcte sur l'autre coté.
Ouvrez les deux button par un double clic et saisissez comme texte ON ou OFF.
D'autre part, vous devrez choisir SELECTABLE et RADIO BUTN.
Ensuite, copiez toutes la construction en nombre suffisant.
Le dernier élément important est un autre button auquel vous donnerez le texte 'OKAY!' et que vous définirez par :
SELECTABLE, DEFAULT et EXIT.
La structure est alors créée, excepté le titre qui est constitué d'une grosse boite avec trois éléments du type TEXT.
Maintenant, nous devons créer les références afin que notre programme sache par la suite quelle boite choisir. Pour cela, il faut utiliser la fonction NAME qui se trouve dans le menu OPTION.
Bien entendu, il faudra ne donner un nom qu'aux objets qui auront une fonction dans le programme. Dans notre cas, nous avons choisi les noms : ELITEON, ELITEOFF, PROPON, PROPOFF, ITALEON, ITALOFF, CONDENON, CONDENOF, MICROON, MICROOFF, SKIPON, SKIPOFF, RAND0, RAND 10
et EXIT pour le champ OKAY!.
Revenez au menu-file et laissez le RCS sauvegarder le tout sous le nom 'PRINIT.RSC' avec l'option SAVE As.

Cliquez sur la ligne de fin de la fenêtre inférieure jusqu'à ce que le RCS ait créé le fichier désiré (la view-window est alors vide de nouveau).
Sur votre disquette vous trouverez les fichiers suivants :
PRINIT.RSC- resourcefile pour le programme suivant
PRINIT.H- fichier include avec des constantes symboliques
PRINIT.DEF- un fichier aide RCS.

Maintenant que nous vous avons expliqué l'utilisation du RCS, nous allons vous donner des informations sur les éléments existant et leur paramètrage éventuel.

Il y a tout d'abord le type object. La plupart des objets qui sont utilisés pour la construction de vos ressources, seront des boîtes :

G_IBOX, G BOX boîtes vides
G_BOXCHAR contient un seul caractère

Si vous devez relier un texte d'une forme quelconque au fichier RSC, vous pouvez utiliser les text-objects qui suivent et qui peuvent créer tous les caractères normaux compte tenu de la couleur de fond et du caractère :

G_STRING une chaîne de caractères
G_BUTTON une chaîne contenue dans une boite
G_TITLE une chaîne dans une liste de menu

D'autre part, le RCS reconnaît des types de texte formattés qui servent à l'édition de messages (p.ex. dans le menu de sélection de fichier).

G_TEXT est un texte formatté
G_BOXTEXT est un texte formatté dans une boite
G_FTEXT est un texte qui peut être édité
G_FBOXTEXT texte qui peut être édité dans une boite.

Une fois que vous avez introduit les objets désirés dans votre arbre, il faudra décider du statut des objets et positionner quelques flags :

SELECTED	dessine un objet en vidéo inversée
CROSSED	met une croix sur une boite
CHECKED	un signe est mis sur le coté gauche de l'objet
DISABLED	L'objet est affiché avec une intensité moyenne

OUTLINED	L'objet est encadré (ne pas utiliser avec SHADOWED)
SHADOWED	une ombre de la boite définie est dessinée (ne pas utiliser avec OUTLINED)
SELECTABLE	l'objet défini ainsi peut être activé durant le déroulement d'un programme.
DEFAULT	la touche RETURN permet de sélectionner cet objet qui est encadré plus visiblement.
EXIT	termine un dialogue
EDITABLE	l'objet contient un texte que l'on peut éditer
RBUTTON	l'objet fait partie d'un groupe dans lequel on peut choisir un seul objet à chaque fois.
HIDETREE	l'objet ainsi désigné n'est pas dessiné par un appel OBJC_DRAW.
TOUCHEXIT	Dès que le pointeur de souris se trouve sur un tel objet, le dialogue est terminé (même sans cliquer).

Dans les pages qui suivent, vous trouverez le listing du programme correspondant. Il faut remarquer que le fichier .H est relié au fichier principal à l'aide d'une fonction d'édition.
Si vous avez créé votre propre RSC-file à l'aide du Resource Construction Set, il faudra faire attention à copier le numéro de chaque symbole. Voyez votre propre .H-file et mettez y à la place votre propre valeur!

```
#define FXMENU 0          /* TREE */
#define EXIT 5            /* OBJECT in TREE #0 */
#define ELITEEIN 7        /* OBJECT in TREE #0 */
#define PROPEIN 10        /* OBJECT in TREE #0 */
#define PROPAUS 11        /* OBJECT in TREE #0 */
#define ITALEIN 13        /* OBJECT in TREE #0 */
#define ITALAUS 14        /* OBJECT in TREE #0 */
#define CONDENEI 16       /* OBJECT in TREE #0 */
#define CONDENAU 17       /* OBJECT in TREE #0 */
#define MICROEIN 19       /* OBJECT in TREE #0 */
#define MICROAUS 20       /* OBJECT in TREE #0 */
#define SKIPEIN 22        /* OBJECT in TREE #0 */
#define SKIPAUS 23        /* OBJECT in TREE #0 */
#define RANDO 26          /* OBJECT in TREE #0 */
#define RAND10 25         /* OBJECT in TREE #0 */
#define ELITEAUS 8        /* OBJECT in TREE #0 */
```

```
/*********************************************************************/
/**************              PROGRAMME: PR-INIT              *************/
/********  ACCESORY d'installation de l'imprimante sur le port parallèle*****/
/**************      (c) J. Walkowiak, 4. Novembre 1985        *************/
/*********************************************************************/
#include "obdefs.h"                 /* définitions des objets              */
#include "gemdefs.h"                /* Definitiones pour GEM               */
#include "define.h"
#include "gembind.h"
#include "vdibind.h"

/*********************************************************************/
/*                                  Definitionen pour RSC-File          */
/*********************************************************************/
#define FILENAME "PRINIT.RSC"       /* nom du fichier RSC                  */
#define MAX_DEPTH 34                /* Nbre d'objets, profondeur des caract. */
#define FXMENU 0                    /* TREE                                */
#define EXIT 5                      /* OBJECT in TREE #0                   */
#define ELITEEIN 7                  /* OBJECT in TREE #0                   */
#define ELITEAUS 8                  /* OBJECT in TREE #0                   */
#define PROPEIN 10                  /* OBJECT in TREE #0                   */
#define PROPAUS 11                  /* OBJECT in TREE #0                   */
#define ITALEIN 13                  /* OBJECT in TREE #0                   */
#define ITALAUS 14                  /* OBJECT in TREE #0                   */
#define CONDENEI 16                 /* OBJECT in TREE #0                   */
#define CONDENAU 17                 /* OBJECT in TREE #0                   */
#define MICROEIN 19                 /* OBJECT in TREE #0                   */
#define MICROAUS 20                 /* OBJECT in TREE #0                   */
#define SKIPEIN 22                  /* OBJECT in TREE #0                   */
#define SKIPAUS 23                  /* OBJECT in TREE #0                   */
#define RAND0 26                    /* OBJECT in TREE #0                   */
#define RAND10 25                   /* OBJECT in TREE #0                   */

long menu_tree;                     /* adresse de l'objet RSC désiré       */

/*********************************************************************/
/*                      Definitions des types de BUTTON dans les menus    */
/*********************************************************************/
#define SELECTED 0x0001
#define NORMAL 0x0000
```

```
/*****************************************************************************/
/*                              code de controle de l'imprimante             */
/*                                   ici : EPSON FX-80+                      */
/*****************************************************************************/
#define RET 13                      /* Return                                */
#define ESC 27                      /* Escape                                */
#define SCHMAL 15                   /* minuscules                            */
#define SCHMALOFF 18
#define ELITE 77                    /* Elite                                 */
#define ELITEOFF 80
#define PROPORTIONAL 112            /* mode élargi        :                  */
#define PSET 1                      /*  on                                   */
#define PRESET 0                    /*  off                                  */
#define ITALIC 4                    /* écriture italique                     */
#define ITALICOFF 5
#define MICRO1 83                   /* Superscript1                          */
#define MICRO2 0
#define MICROOFF 84
#define SKIP 78                     /* Skip over Perforation                 */
#define SKIPOFF 79
#define LRAND 108                   /* mettre marge gauche                   */
#define LAUSSEN 0                   /*  vers la gauche                       */
#define POS10 10                    /*  imprime à partir de position 10      */


#define NO_WINDOW (-1)

#define MIN_WIDTH  (2*gl_wbox)
#define MIN_HEIGHT (3*gl_hbox)

/*****************************************************************************/
/*                                variables globales                         */
/*****************************************************************************/
int contrl[12];                     /* vecteurs de controle                  */
int intin[128];
int ptsin[128];
int intout[128];
int ptsout[128];                    /* suffisamment de place pour tous les cas*/
int pxyarray[12];                   /* vecteur pour coordonnées x,y           */
```

```
int work_in[11];                    /* saisie dans vecteur GSX                */
int work_out[57];                   /* sortie du vecteur GSX                  */


int handle,i;                      /* virtual workstation handle             */
int phys_handle;                   /* physical workstation handle            */
int wi_handle;                     /* Window handle                          */

extern gl_apid;                    /* reconnaissance de l'application        */
extern long gemdos();              /* pour   GEMDOS-Call                     */

int menu_id;                       /* recon. de l'acessory dans deskmenu  */

int gl_hchar, gl_wchar;            /* hauteur et largeur du caractère        */
int gl_wbox, gl_hbox;

int xwork,ywork,wwork,hwork;       /* taille de la fenètre                   */
int xdesk,ydesk,wdesk,hdesk;       /* taille du Desktop                      */
int xold, yold, hold, wold;        /* var. d'aide pour manip. des fenètres   */
int xobj,yobj,wobj,hobj;           /* taille d'un objet                      */
int mausx, mausy;                  /* où est la souris,quand qqch. se passe? */

int dummy;                         /* ... pour paramètres supplémentaires    */

int event;                         /* quel périph. indique quoi              */
int msgbuff[8];
int title, item;                   /* titre du menu et objet courant         */

int ende;


int   top_window;    /* handle of topped window */

int   keycode;    /* keycode returned by event-keyboard */
int   mx,my;       /* mouse x and y pos. */
int   butdown;    /* button state tested for, UP/DOWN */
int   ret;       /* dummy return variable */
```

```
int   hidden;       /* current state of cursor */

int   fulled;       /* current state of window */


/*****************************************************************/
/* open virtual workstation                  */
/*****************************************************************/
open_vwork()
{
int i;
   for(i=0;i<10;work_in[i++]=1);
   work_in[10]=2;
   handle=phys_handle;
   v_opnvwk(work_in,&handle,work_out);
}


/*****************************************************************/
/* open window                           */
/*****************************************************************/
open_window()
{
   wi_handle=wind_create(0x0000,xobj,yobj,wobj,hobj);
                                 /* fenètre aussi grosse que  Dialogbox(obj)*/
/*   wind_set(wi_handle, WF_NAME," le nom est ici ",0,0);  seulement si
fenètre avec ligne de titre
   graf_growbox(xdesk+wdesk/2,ydesk+hdesk/2,gl_wbox,gl_hbox,xdesk,ydesk,wdesk,hdesk);*/
   wind_open(wi_handle,xobj,yobj,wobj,hobj);
                                 /* ouvrir fenètre de travail                */
   wind_get(wi_handle,WF_WORKXYWH,&xwork,&ywork,&wwork,&hwork);
}
```

```
/*****************************************************************/
/*      Accessory Init. Until First Event_Multi      */
/*****************************************************************/
main()
{
   appl_init();
   phys_handle=graf_handle(&gl_wchar,&gl_hchar,&gl_wbox,&gl_hbox);
   menu_id=menu_register(gl_apid,"  FX-80+ INIT");
   wind_get(0, WF_WORKXYWH, &xdesk, &ydesk, &wdesk, &hdesk);

      if(!rsrc_load(FILENAME))      /* charge fichiers RSC                    */
         {
         form_alert(1,"[3][Pirate ou quoi?|PRINIT.RSC|demeure
introuvable.][Abbruch]");
         }
      if(rsrc_gaddr(0,0,&menu_tree)== 0)
         {
         form_alert(1,"[3] [erreur grave!|Resource File pas OK.][Abbruch]");
         }

      rsrc_gaddr(R_TREE,FXMENU,&menu_tree);
      form_center(menu_tree,&xobj,&yobj,&wobj,&hobj);

   multi();
}

multi()
{
   while (TRUE) {
      event = evnt_multi (MU_MESAG | MU_BUTTON | MU_KEYBD,
         1,1,1,
         0,0,0,0,0,
         0,0,0,0,0,
         msgbuff,0,0,&mausx,&mausy,&dummy,&dummy,
         &dummy,&dummy);
```

```
    if (event & MU_MESAG)
       switch (msgbuff[0]) {

          case AC_OPEN:
             if (msgbuff[4] == menu_id) {
                open_vwork();
                open_window();
                output();
                wind_close(wi_handle);
                wind_delete(wi_handle);
                v_clsvwk(handle);
             }
             break;
        }   /* switch */
    } /*while TRUE */
}

output()
{
      rsrc_gaddr(R_TREE,FXMENU,&menu_tree);
      form_center(menu_tree,&xobj,&yobj,&wobj,&hobj);
      form_dial(0,xobj,yobj,wobj,hobj);
      form_dial(1,1,1,1,1,xobj,yobj,wobj,hobj);

      objc_draw(menu_tree,0,MAX_DEPTH,0,0,wdesk,hdesk);

      ende = FALSE;                    /* sinon, ça ne marche qu'une fois          */
      while (ende != TRUE){
          event=evnt_button(1,1,1,&mausx,&mausy,&dummy,&dummy);
          item=objc_find(menu_tree,FXMENU,13,mausx,mausy);
                                       /* quel objet dans menu_tree à pos. souris */

          switch(item){
               case ELITEEIN:
               objc_change(menu_tree,ELITEEIN,0,xobj,yobj,wobj,hobj,SELECTED,1);
               objc_change(menu_tree,ELITEAUS,0,xobj,yobj,wobj,hobj,NORMAL,1);
               gemdos(0x5,ESC);
               gemdos(0x5,ELITE);
               break;
```

```
case ELITEAUS:
objc_change(menu_tree,ELITEAUS,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,ELITEEIN,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,ELITEOFF);
break;

case CONDENEI:
objc_change(menu_tree,CONDENEI,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,CONDENAU,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,SCHMAL);
break;

case CONDENAU:
objc_change(menu_tree,CONDENAU,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,CONDENEI,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,SCHMALOFF);
break;

case PROPEIN:
objc_change(menu_tree,PROPEIN,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,PROPAUS,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,PROPORTIONAL);
gemdos(0x5,PSET);
break;

case PROPAUS:
objc_change(menu_tree,PROPAUS,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,PROPEIN,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,PROPORTIONAL);
gemdos(0x5,PRESET);
break;
```

```
case ITALEIN:
objc_change(menu_tree,ITALEIN,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,ITALAUS,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,ITALIC);
break;

case ITALAUS:
objc_change(menu_tree,ITALAUS,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,ITALEIN,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,ITALICOFF);
break;

case MICROEIN:
objc_change(menu_tree,MICROEIN,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,MICROAUS,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,MICRO1);
gemdos(0x5,MICRO2);
break;

case MICROAUS:
objc_change(menu_tree,MICROAUS,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,MICROEIN,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,MICROOFF);
break;

case SKIPEIN:
objc_change(menu_tree,SKIPEIN,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,SKIPAUS,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,SKIP);
break;
```

```
                case SKIPAUS:
                objc_change(menu_tree,SKIPAUS,0,xwork,ywork,wwork,hwork,SELECTED,1);
                objc_change(menu_tree,SKIPEIN,0,xwork,ywork,wwork,hwork,NORMAL,1);
                gemdos(0x5,ESC);
                gemdos(0x5,SKIPOFF);
                break;

                case RAND0:
                objc_change(menu_tree,RAND0,0,xwork,ywork,wwork,hwork,SELECTED,1);
                objc_change(menu_tree,RAND10,0,xwork,ywork,wwork,hwork,NORMAL,1);
                gemdos(0x5,ESC);
                gemdos(0x5,LRAND);
                gemdos(0x5,LAUSSEN);
                break;

                case RAND10:
                objc_change(menu_tree,RAND10,0,xwork,ywork,wwork,hwork,SELECTED,1);
                objc_change(menu_tree,RAND0,0,xwork,ywork,wwork,hwork,NORMAL,1);
                gemdos(0x5,ESC);
                gemdos(0x5,LRAND);
                gemdos(0x5,POS10);
                break;

                case EXIT:
                objc_change(menu_tree,EXIT,0,xwork,ywork,wwork,hwork,SELECTED,1);
                gemdos(0x5,RET);
                form_dial(2,xobj,yobj,wobj,hobj);
                form_dial(3,1,1,1,1,xobj,yobj,wobj,hobj);
                ende=TRUE;
                objc_change(menu_tree,EXIT,0,xwork,ywork,wwork,hwork,NORMAL,1);
                /* remettre en place, sinon il sera encore activé au prochain
appel d'accesory*/
                break;

            } /* fin switch  */

        } /* fin while */
}
```

Dans le listing précédent, seule une partie des définitions de symboles était peut être intéressante pour vous. C'est peut être le cas si vous ne disposez pas d'une EPSON ou d'une imprimante compatible, car dans ce cas, il faudra adapter les codes de la liste d'initialisation à votre appareil.
Autrement, le listing ne vous aura sans doute pas causé de grand mal de tête car tout y est très clair. Les appels 'graf_growbox' et 'graf_shrinkbox' de la routine window sont des petites transformations qui ne sont en fait que des opérations esthétiques. Elles servent à agrandir ou à réduire rapidement le contour.

L'appel du RSC-file est nouveau. Etant donné que 'rsrc_load()' est une fonction elle donne un résultat qui est soit 'TRUE' soit 'FALSE' (vrai ou faux). Pour cette raison, dans le cas d'une erreur durant le déroulement, un 'alert tree' est affiché et le programme est arrêté.
Le plus important est l'appel 'rsrc_gaddr()' car cette fonction donne le pointeur de l'adresse d'un objet recherché.

Exemple :
Après la mise en route du programme, il faudra sans doute montrer l'arbre complet du premier au dernier objet. Il faut donc savoir où se trouve la racine dans notre cas l'arbre FXMENU) en mémoire. nous appelons donc 'rsrc_gaddr()' et indiquons à l'AES ce que nous recherchons, c'est à dire l'objet FXMENU dans un arbre (R_TREE). L'adresse de FXMENU doit prendre pour valeur le pointeur &menu_tree :

```
rsrc_gaddr(R_TREE,FXMENU,&menu_tree);
```

Pour afficher l'arbre ou bien un objet il faut appeler object_draw qui est une fonction qui dessine un secteur d'un arbre :

```
objc_draw(menu_tree,0,MAX_DEPTH,0,0,wdesk,hdesk);
```

Les paramètres indiquent respectivement quel arbre sera dessiné, à partir de quel objet (ici nul, c'est à dire le premier), jusqu'à

quel objet, au nombre maximum, est lu dans RCS sous le nom INFO) et une surface maximum pour afficher le tout. L'appel Object_Find dans la boucle principale donne le numéro de l'objet se trouvant sous la souris quand on lui indique la position de la souris (qui est donnée par le quatrième et le cinquième paramètre de 'event_ button'). Ce numéro est alors comparé avec la liste de nos numéros d'objets qui sont précisés par les constantes symboliques jusqu'à ce qu'il y ait correspondance. L'action correspondante est alors effectuée.
L'instruction 'object_change' que l'on trouve sert tout simplement à donner la couleur noire à l'objet.
Les instructions form_dial que l'on peut également trouver dans le listing servent à dessiner la dialog box. Elles libèrent la place nécessaire (il faut que la partie d'écran qui se trouve dessous soit sauvegardée), dessinent la boîte agrandie ou rapetissée et rétablissent le secteur effacé.

4.5. PRINIT EN TANT QU'ACCESSORY

Pour installer l'application en tant qu'accessory, il faut réaliser un certain nombre de modifications. Ainsi un accessory utilisera une fenêtre qui ne doit pas être plus grande que la dialogbox. Elle n'est pas pratique lorsque l'accessory doit être déplacé sur l'écran et elle est rarement appelée. Cela nous épargne du travail de programmation et permet de gagner de la place mémoire et du temps de chargement. Dans le secteur 'open_window', 'wind_create' et 'wind_get' sont suffisants. Pour ceux qui désirent absolument avoir une fenêtre vous trouverez dans le listing les lignes nécessaires sous forme de commentaire.

L'aspect important pour rendre un accessory disponible est l'installation dans le desk-menu, un rôle que rempli la fonction 'menu_ register'. Elle utilise comme paramètre d'entrée le code ID de l'application (de appl_init) et une chaîne sous laquelle le programme apparaîtra dans le menu. le résultat est un chiffre entre 0 et 5 dans 'menu_register' qui correspond au code ID de l'accessory.

Pour en arriver là, le programme doit être mis en route, ce qui se passe avec chaque accessory lors du chargement du système d'exploitation. L'utilisateur ne se rend compte de rien car la station de travail n'a pas encore été ouverte.

Après cette phase d'initialisation, l'accessory se branche sur une instruction 'event_multi' qui doit admettre toutes les interruptions pour ne pas perturber le déroulement du programme principal. L'accessory attend un message-event qui laisse l'ID (identificateur) d'accessory dans le tampon de message.

Comme nous l'avons déjà indiqué, le code de l'application à appeler se trouve dans msgbuff(4). Donc quand la condition :

```
if (msgbuff(4)==menu_id)
```

a pour résultat TRUE, l'utilisateur a appelé l'accessory en question.
C'est alors seulement que la station de travail et la fenêtre sont ouvertes et que s'effectue le branchement au programme (ici : output();). Cela conduit à un 'test event' et correspond en général au programme normal. Il faut seulement faire attention à ce que la condition d'arrêt soit toujours FALSE. Cela peut se produire avant de quitter (et pas avant la fin!) le programme ou bien au début de la boucle principale car un nouvel appel de l'accessory est possible.

Avant tout, il faut faire attention à ce que le déroulement d'un accessory ne soit jamais terminé et il ne faut donc jamais trouver un appl_exit. Les accessory fonctionnent toujours en multitâche, c'est à dire que chaque accessory est dans la 'ready list' et le 'test event_multi' d'un accessory est toujours réalisé.
La structure d'un tel appel event_multi est donc :

```
while (TRUE)
        ... event_multi /* test */
        ...Message_event ?
```

```
                    ...si oui propre menu id?
                            ... si oui, on exécute
                    ...si non: on continue d'attendre
            ... si non  on continue d'attendre
../* fin du WHILE */
```

Le code du programme ne doit contenir aucune instruction qui permette d'arrêter cette boucle. La condition est donc TRUE, ce qui est toujours le cas!

```
/***************                PROGRAMME: PR-INIT                ************/
/******** programme d'installation d'imprimante sur port parallèle**********/
/***************       (c) J. Walkowiak, Octobre 1985          ************/
/*****************************************************************************/
#include "obdefs.h"                 /* définition d'objets                   */
#include "gemdefs.h"                /* Definitions pour GEM                  */
#include "define.h"
#include "osbind.h"
#include "gembind.h"
#include "vdibind.h"
#include "portab.h"

/*****************************************************************************/
/*                                 Definitions pour RSC-File                 */
/*****************************************************************************/
#define FILENAME "PRINIT.RSC"       /* nom du fichier RSC                    */
#define MAX_DEPTH 34                /* nbre d'objets, profondeur du caractère*/
#define FXMENU 0                    /* TREE                                  */
#define EXIT 5                      /* OBJECT in TREE #0                     */
#define ELITEEIN 7                  /* OBJECT in TREE #0                     */
#define ELITEAUS 8                  /* OBJECT in TREE #0                     */
#define PROPEIN 10                  /* OBJECT in TREE #0                     */
#define PROPAUS 11                  /* OBJECT in TREE #0                     */
#define ITALEIN 13                  /* OBJECT in TREE #0                     */
#define ITALAUS 14                  /* OBJECT in TREE #0                     */
#define CONDENEI 16                 /* OBJECT in TREE #0                     */
#define CONDENAU 17                 /* OBJECT in TREE #0                     */
#define MICROEIN 19                 /* OBJECT in TREE #0                     */
#define MICROAUS 20                 /* OBJECT in TREE #0                     */
#define SKIPEIN 22                  /* OBJECT in TREE #0                     */
#define SKIPAUS 23                  /* OBJECT in TREE #0                     */
#define RAND0 26                    /* OBJECT in TREE #0                     */
#define RAND10 25                   /* OBJECT in TREE #0                     */

long menu_tree;                     /* Adresse de l'objet RSC choisi         */
```

```
/*****************************************************************************/
/*                        Definitions des types de button dans les menus     */
/*****************************************************************************/
#define SELECTED 0x0001
#define NORMAL 0x0000
#define WI_KIND 0x0001              /* la fenètre n'a que des lignes de noms */
/*****************************************************************************/
/*                                   code de controle de l'imprimante        */
/*                                    ici : EPSON FX-80+                     */
/*****************************************************************************/
#define RET 13                      /* Return                                */
#define ESC 27                      /* Escape                                */
#define SCHMAL 15                   /* minuscules                            */
#define SCHMALOFF 18
#define ELITE 77                    /* Elite                                 */
#define ELITEOFF 80
#define PROPORTIONAL 112            /* écriture élargie                      */
#define PSET 1                      /*  on                                   */
#define PRESET 0                    /*  off                                  */
#define ITALIC 4                    /* italique                              */
#define ITALICOFF 5
#define MICRO1 83                   /* Superscript1                          */
#define MICRO2 0
#define MICROOFF 84
#define SKIP 78                     /* Skip over Perforation                 */
#define SKIPOFF 79
#define LRAND 108                   /* mettre en place marge gauche          */
#define LAUSSEN 0                   /*  vers la gauche                       */
#define POS10 10                    /*  imprime à partir de la position 10   */


/*****************************************************************************/
/*                                   variables globales                      */
/*****************************************************************************/
int contrl[12];                     /* vecteur de controle                   */
int intin[128];
int ptsin[128];
int intout[128];
int ptsout[128];                    /* place nécessaire pour tous les cas    */
int pxyarray[12];                   /* vecteur pour coordonnées x, y         */
```

```
int int_in[11];                    /* saisie dan vecteur GSX                */
int int_out[57];                   /* sortie du vecteur GSX                 */


int handle,i;                      /* virtual workstation handle            */
int phys_handle;                   /* physical workstation handle           */
int wi_handle;                     /* Window handle                         */

int ap_id;                         /* reconnaissance de l'application       */

int gl_hchar, gl_wchar;            /* hauteur et largeur du caractère       */
int gl_wbox, gl_hbox;

int xwork,ywork,wwork,hwork;       /* taille de la fenètre                  */
int xdesk,ydesk,wdesk,hdesk;       /* taille du desktop                     */
int xold, yold, hold, wold;        /* variables d'aide pour manip. fenètres */
int xobj,yobj,wobj,hobj;           /* taille d'un objet                     */
int mausx, mausy;                  /* où est la souris si un évennt survient*/

int dummy;                         /* ... pour paramètres supplémentaires   */

int event;                         /* quel périph. indique quoi              */
int title, item;                   /* titre du menu et objet courant        */

/*************************************************************************/
/*                                ouvrir fenètre et fermer               */
/*************************************************************************/
open_window()
{
   wi_handle=wind_create(WI_KIND,xdesk,ydesk,wdesk,hdesk);
   graf_growbox(xdesk+wdesk/2,ydesk+hdesk/2,gl_wbox,gl_hbox,xdesk,ydesk,wdesk,hdesk);
   wind_open(wi_handle,xdesk,ydesk,wdesk,hdesk);
   wind_get(wi_handle,WF_WORKXYWH,&xwork,&ywork,&wwork,&hwork);
}

close_window()
{
   wind_close(wi_handle);
   graf_shrinkbox(xwork+wwork/2,ywork+hwork/2,gl_wbox,gl_hbox,xwork,ywork,wwork,hwork);
   wind_delete(wi_handle);
}
```

```
open_vwork()
{
int i;
      for (i = 1; i <10; i++){
        int_in[i] = 1;              /* init int_in array: linetype, couleur, */
        }                           /* fillstyles etc.                       */
      int_in[10] = 2;               /* utilise coordonnées RC                */
      handle=phys_handle;
      v_opnvwk(int_in, &handle, int_out); /* on peut commencer     ...       */
}


/*************************************************************************/
/*                                programme principal                    */
/*************************************************************************/
main()
{
int ende;                           /* TRUE si EXIT cliqué                   */
long  gemdos();                     /* pour gemdos-call                      */

      ap_id=appl_init();            /* initialise    GEMAES Array-Structures */

      phys_handle=graf_handle(&gl_wchar,&gl_hchar,&gl_wbox,&gl_hbox);
                                    /* fixe paramètres du desktop            */
      wind_get(0,WF_WORKXYWH,&xdesk,&ydesk,&wdesk,&hdesk);
      open_vwork();                 /* ouvrir workstation                    */
      if(!rsrc_load(FILENAME))      /* charger fichier RSC                   */
         {
         form_alert(1,"[3][pirate, ou quoi  ?je ne trouve pas PRINIT.RSC
         .][Abort]");
         close_window;
         desktop();
         }
      if(rsrc_gaddr(0,0,&menu_tree)== 0)
         {
         form_alert(1,"[3] [erreur grave   !!Resource File pas   OK.][Abort]");
         close_window;
         desktop();
         }
```

```
rsrc_gaddr(R_TREE,FXMENU,&menu_tree);
form_center(menu_tree,&xobj,&yobj,&wobj,&hobj);
form_dial(0,xobj,yobj,wobj,hobj);
form_dial(1,1,1,1,1,xobj,yobj,wobj,hobj);

objc_draw(menu_tree,0,MAX_DEPTH,0,0,wdesk,hdesk);

graf_mouse(3,&dummy);          /* souris=main                              */

while (ende != TRUE){
     event=evnt_button(1,1,1,&mausx,&mausy,&dummy,&dummy);
                               /* attente  cliquage avec bouton gauche  */

     item=objc_find(menu_tree,FXMENU,13,mausx,mausy);
                               /* quel objet dans menu_tree à pos souris*/

     switch(item){
          case ELITEEIN:
          objc_change(menu_tree,ELITEEIN,0,xwork,ywork,wwork,hwork,SELECTED,1);
          objc_change(menu_tree,ELITEAUS,0,xwork,ywork,wwork,hwork,NORMAL,1);
          gemdos(0x5,ESC);
          gemdos(0x5,ELITE);
          break;

          case ELITEAUS:
          objc_change(menu_tree,ELITEAUS,0,xwork,ywork,wwork,hwork,SELECTED,1);
          objc_change(menu_tree,ELITEEIN,0,xwork,ywork,wwork,hwork,NORMAL,1);
          gemdos(0x5,ESC);
          gemdos(0x5,ELITEOFF);
          break;

          case CONDENEI:
          objc_change(menu_tree,CONDENEI,0,xwork,ywork,wwork,hwork,SELECTED,1);
          objc_change(menu_tree,CONDENAU,0,xwork,ywork,wwork,hwork,NORMAL,1);
          gemdos(0x5,SCHMAL);
          break;
```

```
case CONDENAU:
objc_change(menu_tree,CONDENAU,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,CONDENEI,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,SCHMALOFF);
break;
case PROPEIN:
objc_change(menu_tree,PROPEIN,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,PROPAUS,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,PROPORTIONAL);
gemdos(0x5,PSET);
break;
case PROPAUS:
objc_change(menu_tree,PROPAUS,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,PROPEIN,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,PROPORTIONAL);
gemdos(0x5,PRESET);
break;
case ITALEIN:
objc_change(menu_tree,ITALEIN,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,ITALAUS,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,ITALIC);
break;
case ITALAUS:
objc_change(menu_tree,ITALAUS,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,ITALEIN,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,ITALICOFF);
break;
```

```
case MICROEIN:
objc_change(menu_tree,MICROEIN,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,MICROAUS,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,MICRO1);
gemdos(0x5,MICRO2);
break;

case MICROAUS:
objc_change(menu_tree,MICROAUS,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,MICROEIN,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,MICROOFF);
break;

case SKIPEIN:
objc_change(menu_tree,SKIPEIN,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,SKIPAUS,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,SKIP);
break;

case SKIPAUS:
objc_change(menu_tree,SKIPAUS,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,SKIPEIN,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,SKIPOFF);
break;

case RAND0:
objc_change(menu_tree,RAND0,0,xwork,ywork,wwork,hwork,SELECTED,1);
objc_change(menu_tree,RAND10,0,xwork,ywork,wwork,hwork,NORMAL,1);
gemdos(0x5,ESC);
gemdos(0x5,LRAND);
gemdos(0x5,LAUSSEN);
break;
```

```
        case RAND10:
        objc_change(menu_tree,RAND10,0,xwork,ywork,wwork,hwork,SELECTED,1);
        objc_change(menu_tree,RAND0,0,xwork,ywork,wwork,hwork,NORMAL,1);
        gemdos(0x5,ESC);
        gemdos(0x5,LRAND);
        gemdos(0x5,POS10);
        break;

        case EXIT:
        objc_change(menu_tree,EXIT,0,xwork,ywork,wwork,hwork,SELECTED,1);
        gemdos(0x5,RET);
        form_dial(3,xobj,yobj,wobj,hobj);
        form_dial(2,1,1,1,1,xobj,yobj,wobj,hobj);
        ende=TRUE;
        break;

    } /* fin  switch  */

  } /* fin  while */
  desktop();

} /* fin  main() */

desktop()
{
   v_clsvwk();
   appl_exit();
}
```

ENTRETIEN ET REPARATION DU VC 1541
Plus de 250 pages.

Ce livre vous permettra de réparer et d'entretenir le lecteur de disquettes Commodore VC 1541. Vous y trouverez amplement décrits les principes mécaniques et électroniques de ce périphérique. Pour ceux qui ne connaissent pas l'électronique cet ouvrage en est aussi un parfait apprentissage.
Courant Octobre.

Réf. : ML134
Prix : 149 FF

LIVRES DIVERS

ATARI ST TRUCS et ASTUCES
Plus de 250 pages.

Un recueil complet de trucs et d'astuces que votre tout nouveau ATARI ST va beaucoup apprécier ! Des graphismes fantastiques à partir de programmes en BASIC, des exemples et des conseils pour programmer en langage C et en Assembleur...
Disponible courant Novembre.

Réf. : ML140
Prix : 149 FF

La BIBLE de l'ATARI ST
Plus de 500 pages.

Ce livre contient un ensemble complet d'informations sur l'ATARI ST, la description HARDWARE de la machine ainsi que des schémas détaillés et amplement expliqués notamment des interfaces V24, du port d'extension, de l'interface midi, la structure des graphiques, du BIOS, de GEM, les adresses systèmes importantes, le fonctionnement de la souris....
Disponible courant Novembre.

Réf. : ML142
Prix : 249 FF

LE LIVRE DU LANGAGE MACHINE DE l'ATARI ST
Plus de 250 pages.

Tout ce qu'il faut savoir pour tirer au mieux parti de votre ATARI ST : système de calcul et de bit manipulation du 68000 , utilisation des registres, structure des commandes, programmation structurée : récursion, piles, procédures et fonctions, listings sources de programmes assembleurs, routines systèmes... Un super livre !
Disponible Octobre.

Réf. : ML141
Prix : 149 FF

MICRO APPLICATION MICRO APPLICATION

LE LIVRE DU GEM SUR ATARI ST
Plus de 350 pages.

Cet ouvrage contient ce qu'il est nécessaire de savoir pour utiliser GEM efficacement : Fonctionnement de la souris, Virtual Device Interface, Application Environment Services, Graphics Device Operating System. Description de routines utilisant GEM en BASIC, C et en Assembleur. Utilisation standard et spécifique de l'operating system.

Réf. : ML139
Prix : 149 FF

APPLE 2 TRUCS ET ASTUCES POUR APPLE 2E, 2+, 2C

Très important : l'ouvrage APPLE 2 TRUCS ET ASTUCES repose sur l'expérience acquise dans le travail avec le 2+, 2E et le nouveau super compact 2C. Les PEEKS et POKES intéressants, les bases de la programmation en assembleur, le graphisme couleur, la structure des masques-écran ne sont que quelques-uns des nombreux thèmes abordés. Un aperçu sur l'utilisation de logiciels écrits pour l'APPLE 2 complète ce nouvel ouvrage que tout possesseur d'APPLE 2 doit se procurer.

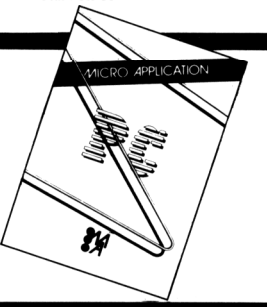
Réf. : ML111
Prix : 149 FF

TRUCS ET ASTUCES POUR TURBO PASCAL

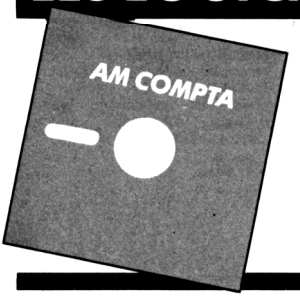
Plus de 250 pages.

Ce livre est un recueil de trucs et d'atuces pour utiliser au mieux Turbo Pascal. Vous trouverez en outre les nombreux conseils et méthodes des programmes exemples et utilitaires comme des procédures de tri, de gestion d'écran ... Un ouvrage très complet indispensable à tout utilisateur de Turbo Pascal.

Réf. : ML133
Prix : 149 FF



LES LOGICIELS AMSTRAD

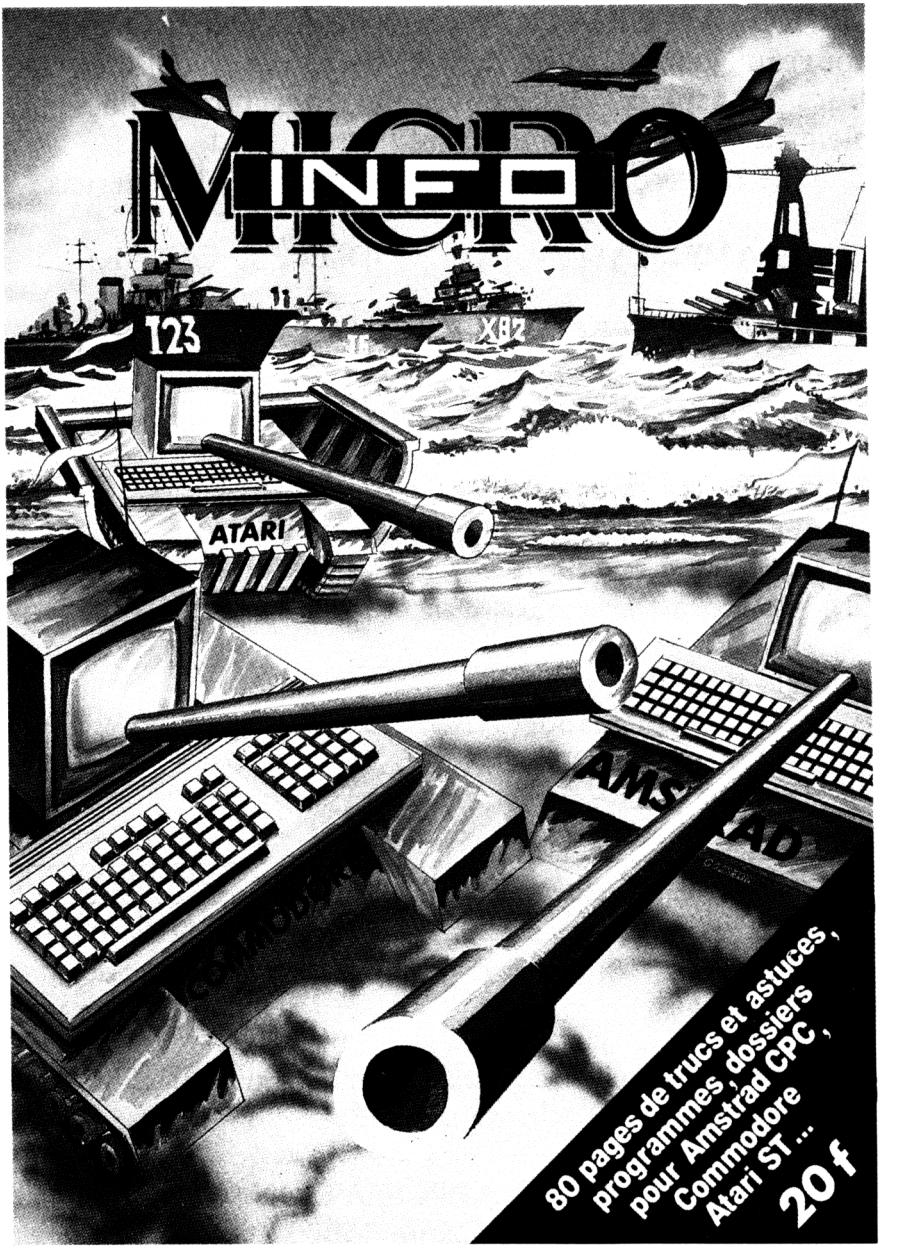


AM COMPTA
pour CPC 664 et 6128 ou 464+DDI

Ce logiciel sur disquette permet au particulier et au professionnel de tenir sa comptabilité à partir de la saisie des recettes et des dépenses. Celles-ci sont imputées dans des postes comptables et ventilées. La consultation permet la visualisation des codes de trésorerie en dépenses / recettes et soldes, des postes comptables en camembert et histogrammes, la recherche à partir du montant ou du numéro de chèque d'une fiche comptable et la consultation des montants non rapprochés. Edition du journal par postes, du grand livre, ventilation par postes et liste des postes.

Réf. : AM307
PRIX : 750 FF

MICRO APPLICATION MICRO APPLICATION


MICRO
INFO
123
ATARI
80 pages de trucs et astuces, programmes, dossiers pour Amstrad CPC, Commodore Atari ST...
20 f

MICRO APPLICATION vous présente MICRO INFO, nouveau journal avec des dossiers, des bidouilles, des trucs et astuces, des nouveautés, des programmes et plein de rubriques sympas! (88 pages)
Chaque numéro traite principalement de 3 matériels:

AMSTRAD - COMMODORE - ATARI

carte d'abonnement

Je désire m'abonner à MICRO INFO

☐ Le numéro 1 : 15 F + 5 F pour frais d'envoi
☐ Le numéro 2 : 20 F + 5 F pour frais d'envoi
☐ Les numéros 1 et 2 : 35 F + 5 F pour frais d'envoi
☐ Je choisis de m'abonner pour 4 numéros au prix de 70F

Je règle par ☐ chèque
☐ mandat
☐ CCP

Nom : _____ Prénom : _____
Adresse : _____
Code postal : _____ date et signature :

Veuillez nous retourner cette carte sous pli ainsi que votre règlement à l'adresse suivante :

MICRO APPLICATION
13 rue Sainte Cécile 75009 PARIS

CE LIVRE VOUS PASSIONNE,

MAIS VOUS N'AVEZ PAS LE TEMPS

DE TAPER LES PROGRAMMES

ACHETEZ LA DISQUETTE

DES PROGRAMMES DU LIVRE

TRUCS ET ASTUCES DE L'ATARI ST

NOM : _____

PRENOM: _____

ADRESSE: _____

_____

CODE POSTAL: _____ **VILLE:** _____

Je désire recevoir la disquette des programmes du livre TRUCS ET ASTUCES DE L'ATARI ST

Je joins à ce coupon un [] chèque de 120 francs.
[] CCP de 120 francs.

Date : **Signature :**

Achevé d'imprimer en mars 1986
sur les presses de l'imprimerie Laballery et Cie
58500 Clamecy
Dépôt légal : mars 1986
N° d'imprimeur : 602090

IMPRIMÉ EN FRANCE

Ce livre est une véritable mine d'or pour tous les heureux possesseurs d'ATARI ST. Différents domaines sont couverts et tous les programmes (écrits en C, Assembleur ou ST Basic) sont clairement commentés.

Principaux thèmes abordés :

- **BASIC et GEM**
- **Les Commandes VDISYS**
- **BASIC et Langage Machine**
- **Hardcopy automatique**
- **Un RAM DISK pour ATARI ST**
- **Spooler d'imprimante**
- **Boot automatique pour applications sous TOS**
- **Langage Machine et Langage C**
- **Hardcopy Couleur**
- **Anatomie de GEM**
- **Exemple d'applications sous GEM**
- **Et beaucoup d'autres programmes, exemples et utilitaires !**

Réf. : ML 140
Prix : 149 FF

ISBN : 2-86899-043-6